

Boston University

OpenBU

<http://open.bu.edu>

Boston University Theses & Dissertations

Boston University Theses & Dissertations

2022

Towards secure computation for people

<https://hdl.handle.net/2144/46383>

"Downloaded from OpenBU. Boston University's institutional repository."

BOSTON UNIVERSITY
GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

**TOWARDS SECURE COMPUTATION
FOR PEOPLE**

by

RAWANE ISSA

B.E., American University of Beirut, 2016

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2022

© 2022 by
RAWANE ISSA
All rights reserved

Approved by

First Reader

Mayank Varia, Ph.D.
Associate Professor of Computing and Data Sciences

Second Reader

Ran Canetti, Ph.D.
Professor of Computer Science

Third Reader

Leonid Reyzin, Ph.D.
Professor of Computer Science

Fourth Reader

Gabriel Kaptchuk, Ph.D.
Research Assistant Professor of Computer Science

“I often sit
There in the sea
And dream dreams
And hope hopes
And wish wishes
And lately
As I listen to the wind song
As it dances a beautiful dance
for me
But these moments
Never seem to last too long
Because after the hopes
The dreams, and the wishes
You got the singin’ of the
dancin’ wind
After you and me
And a stolen moment of hap-
piness
After a glimpse
Of the timeless, natural uni-
verse
Moving in effortless effervesce
Starts when rhythm
Moving in effortless effervesce
Starts when beauty
Comes, comes the reality of
today
Grinning it’s all-knowing,
fiendish grin
Knowing everything I say
Everything I feel
Everything I think
Every gesture I make today,
today

Pressing his ugly face against
mine
Staring at me with lifeless eyes
Crumbling away all memories
of yesterday’s dreams
Crumbling away
Crumbling hopes and destroy-
ing wishes
Destroying dreams
Can’t even get close to no-
body no more
No matter how much you try
You can’t get close to nobody
no more
Because today is a killer
And only you can save us Lord
[...]
I never dreamed
I certainly never hoped
That one day
I’d be screamin’
'Bout something my mother
told me I needed
In the beginnin’ [...]
Today, who are you Lord
You are a killer !”

*Excerpt from Nina Simone’s My Sweet
Lord / Today Is a Killer (from Emer-
gency Ward!, 1972), an adaptation of
David Nelson’s poem Today Is a Killer
and George Harrison’s song My Sweet
Lord.*

Acknowledgments

This thesis is a labor of love for all the people who have accompanied me in my journey through the PhD. I hope that my work will help us see better days. If it does not withstand the test of time, then at the very least my love and appreciation for all of you will.

To my fiancé Kinan. For our road through hardship, for our memories and for the many more years together to come. This above else for you. I am infinitely grateful that I was able to experience one of the very few good things that came out of aftermath of the Syrian civil war. Thank you for making a home for me when there was no none.

To my advisor Mayank, for being someone I am proud to work with and call my advisor and friend. You have helped me become the person I am happy to be today. I am grateful for who you are and for accepting me for who I am.

To my sister Hanane, for being the best sibling and friend I could have asked for. I was incredibly lucky when the universe rolled its dice. Thank you for making me feel accepted and loved.

To my closest friend Brahim, for being the family I chose to have. Thank you for sparing no amount of support and love and for being one of the most beautiful souls I know.

To my mother, father and family, for all their love and support. I hope this work is something you can be proud of.

To my thesis committee, Ran, Leo and Gabe for being supportive and inclusive. Thank you for making the end line far smoother than I could have ever have imagined.

To Mayank, Ran, Leo, Gabe, Adam, Alley, Assaf, Peter G., Leonid L. and Azer, thank you for being educators who have broadened my perspective and made cryptography an actualized field that I could enjoy.

To Jacob, for being a kindred spirit since my childhood. We are not going anywhere with our thoughts and identities. The Middle East had better get used to it.

To Elias, for being a wonderful friend and my Saturday gaming session partner. Stop getting into fictional fights Elias and keep on being as wonderful.

To my cat Kirby, for all the joy you have brought me. I hope that in return I can make your life as happy.

To Dave, Brent, Sebastien, Olivier and the folks at Galois and Stephanie S. for the wonderful experience I had during my internship at Galois and the inclusive and supportive space you provided me with.

To Lna, Joey A., Whard, Hassan, Kareem, Lojy, Marc, Jad S. for being a shelter away from the madness of this world.

To Assaf and Irene, for our evenings together reminiscing over a lost home.

To my friends at SAIL, Peter, San, Ira, Marcella, Lucy, Ben, Fredrick and Motun for all the wonderful work we've built together. I am very proud of us.

To my friends in Boston, Bassel, Sarah and Jack for making the city much less austere.

To Bayan, Mohammad, George, Soubra, Nerces, Adel, Baydoun, Jad El K. and Ramzi for being my friends in mischief at AUB. Cheers to all of you.

To Nicolas, Maurice, René and Saadi for the fun times at the hub.

To the Metal Appreciation Society, James, Ryan, Alex B., Joey D., Valentin, Alex M. and everyone else, for rocking on and spreading the doom.

To my friends at BUsec, Pratik, Palak, Marika, Megan, Luowen, Gavin, Islam and Ari for your support and friendship. I hope the day will come when graduate students will receive the pay and resources that they deserve. I hope that you will not have to struggle as I have.

To my AUB mentors F. Zaraket, L. Bazzi, K. Makdesi, S. Agha and T. Khalidi for believing in me during my undergraduate years.

To Kristin Hayter (Lingua Ignota), John Cullen (nellucnhoj), Nina Simone, N. K. Jemisin and the many other artists who have brought me comfort through this harsh journey. When I knocked on the door of the cosmos, you were there to answer my call and say “You are not alone”.

Rawane Issa

Ph.D. Student

CS Department

TOWARDS SECURE COMPUTATION FOR PEOPLE

RAWANE ISSA

Boston University, Graduate School of Arts and Sciences, 2022

Major Professor: Mayank Varia, PhD
Associate Professor, Faculty of Computing and Data
Sciences

ABSTRACT

My research investigates three questions: How do we customize protocols and implementations to account for the unique requirement of each setting and its target community, what are necessary steps that we can take to transition secure computation tools into practice, and how can we promote their adoption for users at large? In this dissertation I present several of my works that address these three questions with a particular focus on one of them.

First my work on “Hecate: Abuse Reporting in Secure Messengers with Sealed Sender” designs a customized protocol to protect people from abuse and surveillance in online end to end encrypted messaging. Our key insight is to add pre-processing to asymmetric message franking, where the moderating entity can generate batches of tokens per user during off-peak hours that can later be deposited when reporting abuse. This thesis then demonstrates that by carefully tailoring our cryptographic protocols for real world use cases, we can achieve orders of magnitude improvements over prior works with minimal assumptions over the resources available to people.

Second, my work on “Batched Differentially Private Information Retrieval” contributes a novel Private Information Retrieval (PIR) protocol called DP-PIR that is

designed to provide high throughput at high query rates. It does so by pushing all public key operations into an offline stage, batching queries from multiple clients via techniques similar to mixnets, and maintain differential privacy guarantees over the access patterns of the database.

Finally, I provide three case studies showing that we cannot hope to further the adoption of cryptographic tools in practice without collaborating with the very people we are trying to protect. I discuss a pilot deployment of secure multi-party computation (MPC) that I have done with the Department of Education, deployments of MPC I have done for the Boston Women’s Workforce Council and the Greater Boston Chamber of Commerce, and ongoing work in developing tool chain support for MPC via an automated resource estimation tool called Carousels.

Contents

1	Introduction	1
1.1	Hecate: Accountability and Privacy for Abuse Reporting	4
1.2	DP-PIR: Private Information Retrieval for High query loads	9
1.3	Transitioning Cryptographic Tools into Practice	14
1.3.1	Private Evidence Based Policymaking at the Department of Education	14
1.3.2	Web-MPC and JIFF: MPC for social welfare	15
1.3.3	Carousels: Tool chain support for emerging cryptography	16
1.4	Organization	17
2	Hecate: Abuse Reporting in Secure Messengers with Sealed Sender	18
2.1	Introduction	18
2.1.1	Prior work	19
2.1.2	Our contributions	22
2.2	Overview	25
2.2.1	Setting and Threat Model	25
2.2.2	Security goals	28
2.2.3	Protocol Overview	29
2.3	Definitions	33
2.4	Definitions of Cryptographic Building Blocks	33
2.4.1	Modeling an EEMS	37
2.4.2	Defining AMF with Preprocessing	39

2.5	Constructing Hecate	42
2.6	Security Analysis	46
2.6.1	Deniability	48
2.6.2	Anonymity	54
2.6.3	Message Confidentiality and Forward Secrecy	62
2.6.4	Unforgeability and Accountability	66
2.6.5	Backward Security	72
2.7	Implementation and Evaluation	82
2.7.1	Performance Cost and Comparison	83
2.7.2	End-to-End Prototype Deployment	87
2.8	Conclusion and Discussion	88
2.8.1	Extensions	88
2.8.2	Limitations	89
2.8.3	Future Work	90
3	Batched Differentially Private Information Retrieval	92
3.1	Introduction	92
3.2	Motivation	96
3.3	Protocol Overview	104
3.3.1	Setting	105
3.3.2	Threat Model	107
3.3.3	Interpreting Privacy	108
3.4	Incremental Non-Malleable Secret Sharing	110
3.5	Our DP-PIR Protocol	114
3.6	Scaling and Parallelization	135
3.7	Evaluation	137
3.8	Related Work	143

3.9	Conclusion	145
4	Cryptographic Tools for a Wider Audience	148
4.1	Private Evidence Based Policy Making at the Department of Education	149
4.1.1	Background	149
4.1.2	Our Work and Design Goals	150
4.1.3	Practicality and Usability	151
4.1.4	Correctness and Reproducibility	152
4.1.5	Protocol Choice and Performance	152
4.2	Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities	154
4.2.1	Background	155
4.2.2	Web-MPC: Our Work and Design Goals	155
4.2.3	Practicality and Usability	156
4.2.4	Correctness and Error Handling	157
4.3	Carousels: Tool chain support for emerging cryptography	159
4.3.1	Background	159
4.3.2	Motivating Example: Battleships	161
4.3.3	Carousels: Our results	165
5	Conclusion	167
	Bibliography	169
	Curriculum Vitae	194

List of Tables

2.1	A comparison of features and security properties provided by the Signal EEMS protocol as well as several abuse reporting constructions. Security properties are described in §2.2.2 and §2.6. For the anonymity column, \bullet refers to providing anonymity at the level of Signal’s sealed sender [223].	22
2.2	The number of public-key digital signature operations required for each of the interactive algorithms within Hecate (except for the one-time KGen at setup). We only count the additional cryptographic operations required for Hecate beyond those already required by the EEMS. . . .	42
2.3	The format of a franked message delivered to the receiver, along with sizes in bytes for the implementation in §2.7. A franked message sent by the source is similar, except the envelope does not yet contain σ_3 or t_2	49
2.4	Communication overhead of Hecate and other message franking schemes, in bytes.	86
2.5	Runtimes of message franking schemes, in microseconds, for a message size of 1 kB. The runtime of Verify within Peale et al. [202] differs based on whether the message is authored (left) or forwarded (right).	86

2.6	Computation and communication costs for <code>signal-cli</code> with and without <code>Hecate</code> , for a message size of 1 kB. Send and Receive (10^4 trials) correspond to local computation prior to sending or after receiving the message over the network. E2E Latency (600 trials) starts at the beginning of Send and stops at the end of Receive, with network latency included.	87
3.1	Computation and communication complexity of various existing PIR protocols. Here, n is the database size, q^* and q are the number of queries made by a single or different clients. For protocols that support batching, computation complexity represents the total complexity to handle a batch. Communication is always per query	99
3.2	Horizontal scaling with 1M queries and a 100K DB	140
3.3	Expected number of noise queries B per database element as a function of different ϵ (columns) and δ (rows)	140

List of Figures

2·1	Diagram of Hecate’s data flow for a message m , from the source (top) to the forwarder (middle) and then to a reporting receiver (bottom). The commands match our definition of AMF with preprocessing (Def. 11), and the variables <code>token</code> , <code>c_{frank}</code> , <code>c_{stamp}</code> , and <code>report</code> are defined in Fig. 2·2.	23
2·2	The construction of the different parts of a franked cipher. The outputs of the diagram correspond to each party’s contribution to the eventual stamped cipher <code>c_{stamp}</code> . The source constructs the franked cipher <code>c_{frank}</code> using a token provided by the moderator during preprocessing. We denote by <code>payload</code> and <code>envelope</code> two different parts of the ciphertext as defined in the Signal sealed sender protocol [223]; the platform and receiver can read the <code>envelope</code> whereas only the receiver can read the <code>payload</code> .	30
2·3	Hecate’s construction along with the transmissions using the encrypted messenger. The notation $[a \rightarrow b]$ means that party a executes the method and sends the returned value to b . Note that the <code>plat</code> relays messages between the <code>src</code> and the <code>rec</code> . The receiver of a message may elect to forward or report it; we assume that <code>Forward</code> is preceded by a successful invocation of <code>Verify</code> . See Fig. 2·4 for more details on the methods used here.	44
2·4	Hecate’s subroutines. We omit writing out attribute access notation when it is obvious from the context (i.e. <code>com</code> for instance is a shorthand for <code>c_{frank}.com</code>).	45

2·5	Game subroutines and oracles used in several games. Here, d is a fixed parameter known to the moderator.	47
2·6	The security games for Deniability with respect to the Receiver (DENR_b^A) and Moderator (DENM_b^A).	49
2·7	The hybrid steps modifying send_{amf} in the Moderator Deniability game	51
2·8	The hybrid steps modifying send_{amf} in the Receiver Deniability game	53
2·9	The security games for Anonymity with respect to the Receiver and Moderator.	55
2·10	The hybrid steps of send_{amf} in the ANONR_b^A game. We refer the reader to ANONM_b^A hybrids for fwd_{amf} 's hybrids (<i>We omit the final hybrid from this figure for ease of presentation</i>).	57
2·11	The hybrid steps of send_{amf} in the $\text{ANON}_{\text{mod},b}^A$ game. We refer the reader to the moderator anonymity game for fwd_{amf} 's hybrids.	60
2·12	The security games for Message Confidentiality. Here, we use \mathcal{O}^* to denote $\mathcal{O}_{\text{conf-fs},b}^{\text{send}}$, $\mathcal{O}_{\text{conf-fs},b}^{\text{fwd}}$, $\mathcal{O}^{\text{decrypt}}$, $\mathcal{O}^{\text{corrupt}}$ and $\mathcal{O}^{\text{request}}$	62
2·13	The hybrid steps of the CONF-FS_b^A game	64
2·14	The security games for Accountability. Here, we use \mathcal{O}^* to denote $\mathcal{O}^{\text{send}}$, \mathcal{O}^{fwd} , $\mathcal{O}^{\text{deliver}}$, $\mathcal{O}^{\text{request}}$ and $\mathcal{O}^{\text{corrupt}}$	66
2·15	The hybrid steps modifying vfToken in the Accountability game	68
2·16	The security games for Backward Security, where \mathcal{O}^* denotes $\mathcal{O}^{\text{send}}$, \mathcal{O}^{fwd} , $\mathcal{O}^{\text{deliver}}$, $\mathcal{O}^{\text{stamp}}$, $\mathcal{O}^{\text{corrupt}}$ and $\mathcal{O}^{\text{request}}$	73
2·17	The hybrid steps modifying vfMsg and vfCom in the Backward Security game. We use the shorthand <code>token</code> to refer to <code>m_{frank}.payload.token</code> and use the regular expression wildcard operator "*" throughout.	78
2·18	Online runtime of <code>Hecate</code> 's components as a function of message size in bytes	84

2·19	Runtime and dollar pricing of pre-processing token generation (TGen) as function of the token batch size.	85
3·1	Checklist and DP-PIR Total completion time (y-axis, logscale) for varying number of queries (x-axis, logscale) against a 2.5M database .	100
3·2	The ratios of queries/database (y-axis) after which DP-PIR outperforms Checklist by the indicated x factor for different database sizes (x-axis, logscale)	101
3·3	DPF and DP-PIR Total completion time (y-axis, logscale) for varying number of queries (x-axis, logscale) against a 2.5M database	103
3·4	The ratios of queries/database (y-axis) after which DP-PIR outperforms DPF for different database sizes (x-axis, logscale)	104
3·5	SealPIR and DP-PIR Total completion time (y-axis, logscale) for varying number of queries (x-axis, logscale) against a 10K database	105
3·6	The ratios of queries/database (y-axis) after which DP-PIR outperforms SealPIR for different database sizes (x-axis, logscale)	106
3·7	Completion time for varying number of queries against a 100K database (logscale)	138
3·8	Completion time for a batch consisting only of noise queries against varying database sizes (logscale)	139
3·9	Completion time for varying number of parties with 100K queries against a 10K database	139
4·1	Three Secure Battleships Implementation in Oblivious Rust. Syntax is lightly edited for readability.	162
4·2	The online network bandwidth (top, in bits) and rounds (bottom) for the three battleship implementations.	164

List of Abbreviations

AMF	Asymmetric Message Franking
BWWC	Boston Women’s Workforce Council
CCA	Chosen Ciphertext Attack
CPA	Chosen Plaintext Attack
CSAM	Child Sexual Abuse Materials
DP	Differential Privacy
DPF	Distributed Point Function
DP-PIR	Batched Differentially Private Information Retrieval
EEMS	End to End Encrypted Messenger
EUCMA	Existential Unforgeability under Adaptive Chosen-Message attack
E2E	End to End Encryption
FHE	Fully Homomorphic Encryption
GBCC	Greater Boston Chamber of Commerce
MPC	Multi Party Computation
NCES	National Center for Education Statistics
NCMEC	National Center for Missing and Exploited Children
NPSAS	National Post-secondary Student Aid Study
ORAM	Oblivious Random Access Memory
PIR	Private Information Retrieval
PKI	Public Key Infrastructure
PSI	Private Set Intersection
RAM	Random Access Memory
SAIL	Software and Application Innovation Lab
SSN	Social Security Number
U.S.	United States
ZK	Zero Knowledge

Chapter 1

Introduction

Since the 1970s, cryptographers have designed a myriad of protocols and techniques which answer the questions of secure communication, computation, storage and verifiability. More recent years have seen a large number of industrial deployments of private set intersection [144], multi-party computation [67, 109], zero knowledge proofs [108, 258] and a legislative push towards adopting secure techniques in the analysis, sharing and collection of data among (and within) governmental agencies [1, 4, 7]. Some have applauded these efforts and interpreted them as the beginning of a “privacy renaissance” especially within the crypto-currency space [174, 229]. I remain skeptical of the broad claims about the impact of specific technologies [194, 219] and believe that we must reckon as cryptographers with the fact that we haven’t been entirely successful in bringing privacy preserving tools to a wider audience. Have we truly managed to protect people against the growing threat of surveillance capitalism [259]? And if not, who are we conducting our research for? This stance is of course not new [153, 213].

In this dissertation I explore the following questions:

How do we customize protocols and implementations to account for the unique requirements of each setting and its target community, what are the necessary steps that we can take to transition secure computation tools into practice, and how can we promote their adoption for users at large?

This thesis spans 8 published, submitted, and in-progress works that I have compiled

to investigate these questions [8,13,14,24,93,169,170]. Each of my results was carefully designed to address questions of efficiency and practical feasibility, strengthen people's privacy by design, and carefully consider its intended community in the design making process. The solutions I propose are in no way exhaustive but rather record my progress to date.

Motivation. In this section I briefly motivate my approach to address each of the three main questions posed above.

We start with designing and analyzing customized cryptographic protocols to suit emerging needs while maintaining people's existing privacy guarantees. Recent years have witnessed a global reckoning with the devastating impact of misinformation, fake news and online abuse. A heated debate around the ways to combat these campaigns ensued among legislators and governmental agencies on one hand and big tech corporations on the other. At the heart of the discourse is the matter of accountability: How do we hold users liable for illicit information that they share on social media platforms especially when communication channels are end to end encrypted? Many people on both sides of the debate seem to be converging towards the dangerous and misinformed conclusion that accountability is inherently at odds with privacy. Among the community of cryptographers and security experts on the other hand, the conversation has primarily focused on either hampering at ill advised legislation or critiquing proposed technical solutions by big corporations. While both approaches are essential to the matter, neither is sufficient to ensure that people come out of this debate with maximal privacy especially if content moderation becomes legally codified. Our vantage point as experts needs to be one that deters the rise of surveillance, and the core question as a result involves unifying accountability with all existing security properties of private communication. In Chapter 2, I address this question by taking a privacy maximalist approach in a setting where end to end

encrypted communication needs to be extended with accountability.

With regards to efficiency and practical feasibility of secure computation, focusing our research on improving the communication and computation complexity of protocols at scale (e.g., with billions of people) can help further the adoption of privacy tools by more people for several reasons. First, questions of efficiency and scalability can be designed to be non-exclusionary by addressing concerns surrounding the scarcity of computational resources among the most vulnerable people we are trying to protect (e.g. low end phone, or limited data plan). Second, tech companies can be incentivized to adopt privacy tools if their overheads atop existing solutions are minimal. Third, more efficient secure computation can limit tech giants from using any unfounded argument of technical infeasibility vis a vis regulation and efforts towards adopting more privacy preserving tools. To this end in Chapter 3, I present my work on a special-purpose private information retrieval (PIR) construction called DP-PIR, which is targeted specifically at the cases of high query rates in the third chapter.

Finally, we step away from operating in a technical cryptography vacuum and look at the need to interact with our intended community in order to ensure that privacy tools reach a larger and more diverse audience. I believe that most cryptographic work that is motivated by real world settings cannot be conclusive because it is not well equipped to answer the societal facets of a problem like: What are people’s privacy concerns in the settings that we are considering? How do we make privacy accessible and usable to different levels of technical literacy? How do we inform legislators in the policy making process? How do we help developers follow the best security practices? As security experts, we cannot foreshadow what people’s varying privacy needs are and would quickly fall into the fallacy of technological solutionism [10] without outside expertise. The general attitude of the cryptography research community has been to dismiss these types of questions and works as not being technically challenging or

novel. To these claims I go back to the starting point of this thesis: Have we managed to protect people against the threat of surveillance capitalism? And if not, who are we conducting our research for? It is then necessary to refocus the security and privacy lens on the most important party in any computation—people—and inform our decisions regarding all levels of our work (i.e. formal frameworks, protocols and software) from the communities themselves. In Chapter 4, I present some work I have done in this space and highlight the successes that I believe further the adoption of privacy tools in practice.

1.1 Hecate: Accountability and Privacy for Abuse Reporting

The past few years have witnessed a growing discourse on content moderation fueled primarily by the intractable rise of online abuse campaigns [88, 143, 181]. At the heart of the conversation is the apparent tension between on one hand end to end encrypted messaging systems (EEMS) and the privacy guarantees that they provide, and on the other the notion of holding users accountable for messages that they spread [3, 128, 130, 195].

In 2021, Apple responded to this conundrum in the context of combating Child Sexual Abuse Material (CSAM), by announcing a new abuse detection mechanism [34, 51] that would automatically scan user’s messages against a known list of bad content constructed by the National Center for Missing & Exploited Children (NCMEC). There are several specific privacy concerns with Apple’s construction that have been duly highlighted by the information security community [97, 131] such as the possible collisions that could lead to a non-negligible number of false positives, and its failure to provide users with the means to verify the list’s provenance from NCMEC [156, 216].

More generally, I believe that the notion of accountability neither suggests mechanisms which automate the scanning of private messages, nor requires limiting people’s

privacy. In this thesis, I present my work which corroborates my stance and answers:

Is abuse reporting necessarily at odds with the privacy guarantees of end to end (E2E) encrypted messaging systems? How do we empower users to report abuse in a manner that is consistent with E2E principles?

These questions are becoming more vital by the day with the looming legislation that may enforce content moderation mechanisms on EEMS [3, 128, 130, 195].

Our results. In our work on abuse reporting in EEMS, we approach the problem by examining what privacy and security properties of network anonymous secure messengers can be maintained alongside the notion of accountability. It's important to highlight that our work purposefully does not answer the question of whether or not abuse reporting should be implemented in existing EEMS. This broader question is societal in essence and requires a broader array of expertise than the one we present. In this thesis, we focus on the existential question of whether such a system can exist without breaking existing user guarantees of private communication.

At its heart, abuse reporting requires systems to hold users liable for messages that they send. One of the core contributions of Hecate and other abuse reporting schemes [178, 203, 237] is showcasing that accountability does not in fact necessitate automated client or server side scanning.

Our work begins with defining the minimal requirements needed in order to achieve our understanding of this notion. *Accountability*, as defined in Hecate and all prior works in this space, ties the well formed structure of a message to its traceability. In more detail, if a message is to be read by its intended recipient in an abuse reporting scheme extending an EEMS, then it must be possible for it to trace back to its original source. Otherwise it should be considered at the very least equivalent to any other malformed message packet and dropped without ever being displayed to the user.

Notice that nothing in this informal definition necessitates the automated scanning of messages. That is by design: the matter of *when* moderation should happen and *who* should enforce it is semantically distinct from *what* a report is. The added insight of works like Hecate is that accountability can be syntactically/structurally baked into existing EEMS by binding messages to tokens that trace back to the identity of the original source of that message. The philosophical distinction between automated content scanning and abuse reporting lies elsewhere. Abuse reporting takes the position of empowering users by asking them to manually report received content that they deem illicit. The recipient has to pass along the message and its attached token to some moderating entity. Contrast this approach to Apple’s CSAM detection, and other automated message scanning mechanisms, where the user has no control over what list their messages get compared against.

Next, we must address the question of *who* gets to see which parts of a message. In general, reports in these kinds of systems are relayed to a party called the Moderator that is either separate or unified with the underlying EEMS platform/server. Users in the forwarding path of a message are not allowed to read the token/tracing material bound to the message. They are only able to learn their direct interlocutors, especially when a message is forwarded along. Instead, only the moderator is able to decrypt the token and learn the origin of a message in the event of a report.

Notice that we clearly confine accountability to only hold vis a vis the moderator. Another crucial observation of abuse reporting systems like Hecate is that we extend this definition to force the moderator’s knowledge of accountability to be non-transferable, so that the moderator cannot convince anyone else that users sent particular messages. Users can then always repudiate having sent messages to anyone other than the moderator themselves. In doing so, abuse reporting systems can inherit E2E properties such as deniability and confidentiality.

Variations in the literature then stem from four key design decisions: (1) the level of network level anonymity and meta-data leakage provided by the underlying EEMS [203, 237], (2) the content of the report and whether it should only include information related to the originator rather than forwarders of a message [239], (3) the optional separation of the role of the moderator and the platform [203, 237] and (4) the additional functionalities and security guarantees that we may want from these systems [178]. For instance, Asymmetric Message Franking (AMF) [237] provides full network level anonymity at the cost of a computationally intensive protocol that does not allow users to directly forward a message. Source Tracking [203] on the other hand proposes an extremely efficient abuse reporting construction with forwarding that leaks the originator and recipient of a message to the platform acting as a moderator. The Fuzzy Anonymous Complaint Tally System (FACTS) [178] presents even more interesting trade-offs via a construction that only reveals the origin of a message after sufficiently many complaints have been made against it with meta-data leakage akin to Source Tracking.

In Hecate, we instead ensure all the privacy guarantees of the most secure EEMS while maintaining their core functionalities and their efficiency guarantees. We define an abuse reporting scheme that: (1) maintains network level anonymity, (2) allows users to forward messages, (3) guarantees all the properties of EEMS, (4) can separate the roles of moderator and platform and (5) can easily be plugged into any existing EEMS without overly modifying its main functionality. The core insight of Hecate is to introduce a pre-processing stage during which the moderator can create a user-specific batch of tokens that it can later receive at the time of a report. We effectively offload all expensive public operations to an offline stage that can be periodically run during off-peak hours and eliminate the need for the platform to know who the sender of a message is.

Another unique aspect of our work is that we additionally answer the questions of *when* are reports valid with respect to a compromise and what does recovery entail in abuse reporting in EEMS. We define these notions as the duals of the existing properties of backward and forward secrecy in EEMS. We emphasize that forward and backward secrecy for abuse reporting are not trivially inherited from the corresponding properties in the underlying EEMS, because users in an abuse reporting system are held indefinitely accountable for reported messages. Without carefully considering what happens when users recover from an attack in such a system, adversaries may use any retrieved material during an attack to always pin messages on those users. The abuse reporting pieces must therefore eventually become obsolete just as different parts of the underlying EEMS normally are (e.g. via Double Ratcheting [191]). Additionally, we need to thoughtfully introduce these two notions in harmony with all previously mentioned privacy guarantees.

One of the primary motivations of this thesis is to provide practically feasible work, and Hecate is no stranger to the spirit of this dissertation. We provide a construction that satisfies all aforementioned privacy guarantees without compromising on the efficiency of the system. We achieve two goals in this regard: (1) providing a construction that can be easily plugged into an existing EEMS, in our case Signal and (2) showcasing that the computation and communication overhead of using Hecate on top of said EEMS is minimal. We hope that our conclusive experimental results in that regard will incentivize legislators and big corporations not to undermine users' privacy in abuse reporting systems.

For more details, see Chapter 2 for our construction and results.

1.2 DP-PIR: Private Information Retrieval for High query loads

Private Information Retrieval (PIR) is a foundational cryptographic protocol that considers the problem of users who want to privately retrieve data from a remote database. Many variations of this problem have been extensively studied in the literature. However, the literature has largely left an interesting and important subspace of PIR applications unexplored: scenarios where the query load is extremely high. In this introduction, we briefly survey the prior investigations of PIR in order to highlight what is still missing from the picture, and then we describe how our work on Differential Private Batched PIR (DP-PIR) answers some of these shortcomings.

As a motivating example, imagine for instance that a developer wants to check for updates for the software dependencies in their private code-base. Currently, developers do so in the clear and leak information about their library and code-base in the process (e.g. to GitHub). This leakage could include vulnerabilities in the case of outdated or legacy versions, or the functionality of the local code can be inferred from the libraries that it depends on. PIR provides a privacy preserving mechanism by which the user can check for those update without revealing to the server which libraries they are querying for. However, prior PIR protocols cannot efficiently solve this problem.

Background on PIR. The very first PIR protocol was a multi-server information theoretic construction introduced in the late 90s by Chor, Goldreich, Kushilevitz and Sudan (CGKS95) [83]. Several works then improved on the communication and computational complexity of this foundational works. A key obstacle faced the practicality of PIR, namely that the computational server-side work was prohibitively large. Beimel et al. [44] showcased that this limitation, i.e. the $\Omega(n)$ computational

overhead of the server for a database of size n , was in fact inherent to any PIR schemes. Intuitively PIR requires the server to perform work that is linear in the size of the database per query in order to hide any access pattern leakage, namely what the query is/is not. In order to address this shortcoming Beimel et al. proposed two solutions that many works later improved: (1) staging the PIR scheme into an offline/online phase and (2) batching queries to amortize the costs per query.

In the offline/online PIR schemes, the parties partake in a pre-processing phase which handles the bulk of public key operations and results in material that users or servers can later use for a much cheaper online stage. This kind of setting is ideal in real world deployments where resources are more available and are cheaper during certain times of the day. Batching techniques on the other hand utilize coding theoretic techniques (e.g. Batch Codes [146]) to partition the PIR database so that retrieving a batch of entries simultaneously is cheaper than doing otherwise.

Nowadays, practical constructions of PIR rely on these two techniques to achieve sub-linear *online* computation and communication complexity. Most notably, Kogan and Corrigan-Gibbs [162] proposed in 2021 one of the most practical PIR schemes, Checklist, that can feasibly be deployed for block-list checking in the browser (e.g. for certificate revocations). Their overheads are impressively low and can for instance incur no more than a few seconds latency for a database of millions of elements.

Remaining Challenge. However, there are practical caveats in the research on PIR because of how closely the community has stuck to its investigative tradition. In general prior works, both practical and theoretical, have mostly looked at how to lower the online latency without any deliberation to the throughput of the system.

In some sense, these works have managed to overlook the question entirely because they have always treated efficiency as a property of a single isolated query. Consider Checklist for instance. A query in Checklist requires \sqrt{n} online computation, where

n is the size of the table. In an application with a high query rate, where a protocol may need to handle $O(n)$ (or even more) queries over a short period of time, the overall computation becomes super-linear in n .

Real world systems on the other hand are customized for different loads and throughputs in order to carefully allocate valuable resources. For example, databases can be optimized differently based on whether the target load is read or write heavy. In our starting example at the very top of Section 1.2, we can imagine that most GitHub repositories are dependent at the very least on tens of libraries and may be cloned or forked by tens if not hundreds of users. In this kind of scenario, it is essential to design the private updating system for the specific use case where there might be more than 100 times more incoming queries than there are dependencies to check against.

A question hence remains unanswered:

How can we design a practical PIR protocol that is designed for high throughput settings where the number of incoming queries exceeds the size of the database?

Our results. In our work on differentially private multi-server private information retrieval (DP-PIR), we provide a construction that handles a billion queries for databases with millions of entries in a matter of minutes. DP-PIR achieves *constant amortized* communication and computation complexity in the size of the database when the volume of queries is large, as well as constant client work per query. An important insight of DP-PIR is that specializing cryptographic protocols for specific loads can lead to great efficiency gains. In our case, this specialization is intentional and necessary: DP-PIR handles large batches so well for the same reason that it handles small ones poorly. Our construction makes PIR usable in scenarios that were previously impractical or unexplored.

Prior work as I've mentioned have primarily focused on sequentially handling incoming queries. In these kinds of settings, the latency of the system is inversely proportional to its throughput. Meaning that Checklist, DPF [63] and other PIR works that focus on lowering the latency of their protocols, do indirectly improve throughput as a consequence. However, due to the intrinsic lower bounds and overhead that apply to any isolated PIR query, there is only so much that these protocols can do to improve their latency, and thus their throughput. When the number incoming queries q approaches or exceeds the database size n , their complexities grow multiplicatively in q and their isolated query cost, and become prohibitively large (for instance, $O(q\sqrt{n})$ for Checklist).

If we instead opt to batch queries from *different* users, this will no longer necessarily be the case. The latency and throughput are no longer directly related and the PIR lower bounds no longer apply. This is precisely the starting point of DP-PIR. Our work batches queries from multiple users and as a result our server-side communication and computation complexity are $O(n + q)$ and hence additive rather than multiplicative in the number of incoming queries. When $q \gg n$, we effectively amortize these complexities down to a constant amount of work per query.

Our key insight is to pay a fixed overhead independent of the number of queries by providing differential privacy guarantees over the histogram of access patterns of the PIR database: Differential Privacy requires making a bounded number of noisy queries per database entry, while PIR needs the server to touch every element in the database at least once for every query. It is not clear if it is possible to achieve constant amortized complexity when batching queries from different clients without allowing any leakage and our experience indicates that some leakage is almost certainly required. Moreover, revealing a differentially private histogram of access patterns over the database may also be beneficial since it gives servers insight over

how to allocate their resources to accommodate users. In our motivating example for instance, GitHub needs this kind of insight in order to determine which libraries are most susceptible to typo-squatting attacks.

Notice that contrary to the traditional differential privacy problems, our construction trades privacy for efficiency and not accuracy by injecting dummy queries into the system. This may seem unusual but is in fact similar to how mixnets use differential privacy [243] when providing cover traffic. The more noisy queries we introduce into the system, the better privacy guarantees we provide users with, but the slower the system becomes overall. We adjust the noise parameters (i.e. sensitivity, ϵ and δ) according to our desired efficiency needs and the level at which we want the differential privacy guarantees to hold, i.e. at the level of a query, or a user, or all queries a user makes over a window of time. In all of these cases, the total amount of noise injected is much smaller and independent of the number of incoming queries and hence does not constitute a significant performance overhead.

We additionally split DP-PIR into an offline and an online stage, in order to allow users to query the PIR database using only cheap arithmetic “crypto free” operations. Our staging of PIR is similar at high level to prior works where the offline stage handles the bulk of public key operations. One of the differences with previous work is that we require parties to undergo a new offline stage after the previous offline stage material has been consumed. However, this translates to little practical difference because our offline work is also amortized over independent clients, and because existing work invalidates past offline work when the database is updated.

For more details on our construction and results, readers are referred to Chapter 3.

1.3 Transitioning Cryptographic Tools into Practice

In Chapter 4, I describe three projects I have worked on that focus on showcasing the practicality of MPC to a wider and sometimes non-technical audience. In all of these projects, I directly worked during the design making process with the people for whom the cryptographic tools were developed. I believe that this is precisely why these projects were successful.

1.3.1 Private Evidence Based Policymaking at the Department of Education

In 2017, the US Commission on Evidence-Based Policymaking unanimously recommended that inter-agency sharing of administrative data should be accompanied by enhanced privacy protections:

“The Congress and the President should enact legislation establishing the National Secure Data Service (NSDS) to facilitate data access for evidence building while ensuring transparency and privacy. The NSDS should model best practices for secure record linkage and drive the implementation of innovative privacy-enhancing technologies.” [9]

Unfortunately, current approaches for evidence building often involve outsourcing data to third parties that are contractually obligated to safely and securely handle the data while performing agreed-upon computation. In this pilot, we demonstrated to the U.S. Department of Education how multiparty computation can efficiently and securely perform any statistics needed for evidence-based policymaking between agencies with no recourse to anyone outside the department itself and without any privacy risks. This project involved developing the private set intersection construction of Pinkas et al. [206] in a Rust MPC framework called swanky [122], in order to reproduce a portion of the annual 2015–16 National Post-secondary Student Aid Study.

This work was deployed within the usual trust zones of the Department of Education and required as a result making the pilot accessible, usable and understandable to the non-technical people who demonstrated the prototype.

1.3.2 Web-MPC and JIFF: MPC for social welfare

In 2014, the Boston Women’s Workforce Council (BWWC) initiated a city wide study to analyze the wage gap among gender, ethnicity and seniority as part of an effort to advance the interests of women in the workforce. Initially, no third party was willing to undertake the risk of receiving the raw salary data coming from multiple companies partaking in the study. As such, we developed and deployed a web based MPC analytics system for the BWWC to allow them to run their statistics securely [170]. The success of this project was primarily due to our focus on the *accessibility* and *comprehensibility* of the system to a wider audience, and on understanding the various roles and dynamics of participants (e.g., asynchronicity of participation, possible errors in data entry, etc.).

In 2018, the Greater Boston Chamber of Commerce (GBCC) launched the Pacesetters Initiative which aimed to leverage the purchasing power of large and mid-sized companies to create and promote economic opportunities for local minority-owned businesses. In order to assess the progress of the initiative, we later adjusted and deployed our web-based MPC to help the GBCC measure the initiative’s impact on supplier diversity practices, including ways to increase spending with minority-owned businesses. Both the BWWC and GBCC studies are periodic and are run and deployed at least once a year. Later iterations of these deployments used my work on JIFF, a general purpose MPC framework for the web, as a backend.

1.3.3 Carousels: Tool chain support for emerging cryptography

The current landscape of cryptographic protocols consists of a plethora of primitives and problems, each with a space of many solutions customized for specific classes of security, functionality, performance, and application requirements. However, it is often difficult to determine what cryptographic protocol would best fit a particular problem because of the expansive nature of the set of possible solutions and the laborious process of assessing each's possible tradeoffs. For instance, at the macro level, choosing the optimal PIR protocol for a specific application may require evaluating many of the available protocols to identify which one is a good fit. On the other hand, at the micro level, choosing the optimal multiplication protocol or comparison circuit can also be challenging, especially given the number of dimensions that the primitives may provide trade offs in e.g. round vs computation complexity, offline vs online complexity, use of trusted dealers, and many others. In the space of MPC, the problem is exacerbated even further with the construction of hybrid protocols (e.g. ABY, JIFF) that switch between representations in a non-automated way [139]. In short, there is a dire need for tools that help cryptographers and developers navigate the realms of possible solutions according to their dimensions of interest. This is critical for ensuring that continued progress in cryptography can be translated to real world advances. To address this problem, I am currently working on designing and developing Carousels: an automatic performance and resources analysis tool for MPC programs that is language and protocol agnostic and that can reason about different performance metrics, including communication, round, and computation complexity, as well as memory use and cipher sizes. Carousels can be configured to understand and analyze code written in different languages and currently supports analysis of programs written in OblivRust or JIFF. Furthermore, it relies on pluggable cost models that specify how different protocols or primitives behave, including but not limited

to BGW [49], SPDZ [95], Yao [256], and BGV [65].

1.4 Organization

In Chapter 2, I present a formalism and Hecate construction that showcase how accountability can be maintained along side all the privacy guarantees of network anonymous end-to-end encrypted messaging systems. In Chapter 3, I show how to design a Private Information Retrieval protocol called DP-PIR for high throughput use cases. In Chapter 4, I go over my experience with several successful projects that have managed to bring cryptographic tools to a wider audience and some ongoing work I have in this space.

Chapter 2

Hecate: Abuse Reporting in Secure Messengers with Sealed Sender

This chapter is based on joint work [14] with Nicolas Alhaddad and Mayank Varia.

2.1 Introduction

End-to-end encrypted messaging systems like Facebook Messenger, Signal, Telegram, Viber, and WhatsApp are used by billions of people [250] due to their powerful combination of cryptographic protections and ease of use. The security guarantees provided by encrypted messengers are both varied and valuable [241]: confidentiality and integrity from authenticated key exchange [62, 72, 168], deniability from the use of symmetric authenticated encryption [61, 102, 132], and forward and backward (aka post-compromise) security via key evolution [87, 134]. However, these very security guarantees complicate efforts by secure messaging platforms to investigate reports of abuse or disinformation campaigns, which can have serious consequences for individuals and collective society [48, 106, 227, 231, 246].

To address these concerns, the security research community has developed three methods to augment end-to-end messengers with privacy-respecting technologies to assist with content moderation: message franking, source tracing, and automated identification. First, *message franking* [102, 110, 132, 175, 178, 238] allows recipients to manually report abusive messages with assurance that unreported messages retain all guarantees of secure messengers, and reported messages are both accountable (the

moderator correctly identifies the message’s sender) and deniable (the moderator cannot prove this fact to anybody else). Second, *source tracing* [202, 240] allows the moderator to pinpoint the original source of a viral message rather than the person who forwarded the message to the eventual reporter. Finally, *automated identification* [51, 164] proactively matches messages against a moderator-provided list of messages using a private (approximate) set membership test, with possible interventions like rate-limiting or warning labels in case of a match [230]. We refer readers to [217] for more details about content moderation in encrypted settings.

This work contributes a new construction called **Hecate** that simplifies, strengthens, and unifies the first two content moderation techniques: asymmetric message franking and source tracing. We do not consider automated identification, focusing instead on abuse reporting schemes that empower the people who receive messages to choose the action they wish to take [154, 205]. To provide context for our work, we describe the nascent space of message franking and source tracing in more detail before explaining our improvements.

2.1.1 Prior work

There exists a long line of research into the security of end-to-end encrypted messaging systems (EEMS) at both the protocol design and software implementation layers (e.g., [27, 47, 53, 71, 85, 151]). Our work relies on these analyses in order to treat the underlying messaging protocol in a black-box manner and abstract away its details, so we can focus on the additions provided by content moderation protocols.

Message franking constructions involve four parties: a sender and receiver of a message, plus the platform providing the secure messaging service and a moderator who acts on abuse reports (see Figure 2.1). *Symmetric* message franking protocols are limited to the setting in which the platform and moderator are the same entity and have sufficient network-level visibility to pinpoint the sender of each message.

At a high level, these constructions operate as follows: when a sender submits a ciphertext corresponding to the message \mathbf{m} , the platform signs an attestation binding the sender’s identity to a commitment $\text{com}(\mathbf{m})$ provided by the sender in the clear. The receiver also sees this commitment (e.g., as part of a robust encryption scheme [15, 111, 112]) and can check whether it is correct, dropping the packet if it is malformed. Subsequently, the receiver can report the message as abusive by opening all [102, 110, 132] or part [175] of the commitment; then, the platform can determine whether the message is abusive and take appropriate action.

The work of Tyagi et al. [238], which is the starting point for this paper and which we will henceforth refer to as TGLMR, introduces the notion of *asymmetric message franking* (AMF) that removes the limitations from above. Specifically, AMF can operate even when using Signal’s sealed sender [223] or an anonymous communication system (e.g., [25, 90, 101, 244]) that hides the identity of the sender or receiver from the platform. Furthermore, the system is secure whether the moderator and platform are operated by the same or different entities.

Inspired by designated-verifier signatures [149, 215], the TGLMR construction requires the sender to make a Diffie-Hellman tuple $\langle g, g^{sk_{\text{src}}}, g^{k_{\text{mod}}}, g^{sk_{\text{src}} \cdot k_{\text{mod}}} \rangle$ involving the moderator’s secret key and her own, as well as a non-interactive zero knowledge proof that the tuple is well-formed. TGLMR achieves accountability and deniability for the sender, but doesn’t provide forward and backward security due to the use of long-lived secret keys. Moreover, it is complex and expensive to implement (see §2.7), and requires a non-falsifiable knowledge of exponent assumption in the random oracle model. Finally, TGLMR does not easily generalize to more complex conversation graphs that allow for forwarding.

Another line of research investigates the ability for the moderator to trace the original source of messages that might have been forwarded several times within

an EEMS. Tyagi, Miers, and Ristenpart [240] began this line of study with their Traceback scheme, which reveals to the moderator the entire path from the original source to the reporter, but it requires server-side storage proportional to the number of messages eligible to be traced. Two recent works provide *source tracing*, identifying only the original source of a reported message. First, Peale, Eskandarian, and Boneh [202] contribute a source tracing construction that inherits most security properties from the underlying EEMS (see Table 2.1). Using more expensive crypto operations, the stronger variant of their construction is the only one to date to achieve tree unlinkability — namely, that a receiver who gets the same message twice cannot tell if they originate from the same or different sources. Second, the FACTS scheme by Liu et al. [178] provides source tracing along with a threshold reporting scheme so that the moderator only learns when sufficiently many complaints have been lodged against an abusive source client. However, none of these traceback or source tracing schemes [178, 202, 240] considers backward security as part of their security model. Also, none provides full anonymity of senders and receivers from the EEMS platform or moderator; FACTS is compatible with a network that provides one-sided anonymity, but it requires senders to identify themselves and request tokens from the moderator on the fly whenever they wish to send a message.

This leaves the following open question:

Can we design a protocol that simultaneously provides asymmetric message franking (AMF) and source tracing, achieves forward and backward security, maintains anonymity of senders and receivers to the extent provided by the underlying EEMS network, and only makes black-box use of standardized cryptography in the plain model?

In this work, we answer the question in the affirmative.

Construction	Features					Security Guarantees							
	Abuse reporting	Message forwarding	Source tracing	Trace info	Threshold report	Confidentiality	Anonymity	Tree unlinkability	Deniability	Forward security	Backward security	Unforgeability	Accountability
<i>Signal</i>	○	●	×	×	×	●	◐	●	●	●	●	●	×
Tyagi et al. [238]	●	○	×	src	○	●	●	×	●	○	○	●	●
Traceback [240]	●	●	●	path	○	●	○	○	◐	○	○	●	●
FACTS [178]	●	●	●	src	●	●	◐	○	◐	◐	○	●	●
Peale et al. [202]	●	●	●	src	○	●	○	●	●	◐	○	●	●
Hecate (<i>this work</i>)	●	●	●	src	○	●	●	○	●	●	●	●	●

●: fully provided, ◐: provided but not proven, ◑: partially provided, ○: not provided, ×: not applicable

Table 2.1: A comparison of features and security properties provided by the Signal EEMS protocol as well as several abuse reporting constructions. Security properties are described in §2.2.2 and §2.6. For the anonymity column, ◐ refers to providing anonymity at the level of Signal’s sealed sender [223].

2.1.2 Our contributions

In this work, we provide a new definition and construction for asymmetric message franking (AMF) that is more general, more secure, and faster than previous work. To achieve this goal, we revisit the decision by TGLMR [238] to “restrict attention to non-interactive schemes for which franking, verification, and judging requires sending just a single message.” On its face, this restriction seems natural because end-to-end encrypted messengers are designed to work asynchronously in situations with limited network connectivity, so one-round (online) protocols are desirable. However, this restriction also appears to direct the solution space toward expensive crypto tools like designated-verifier signatures and zero knowledge proofs.

Our core insight is to introduce an *AMF with preprocessing* model as shown in

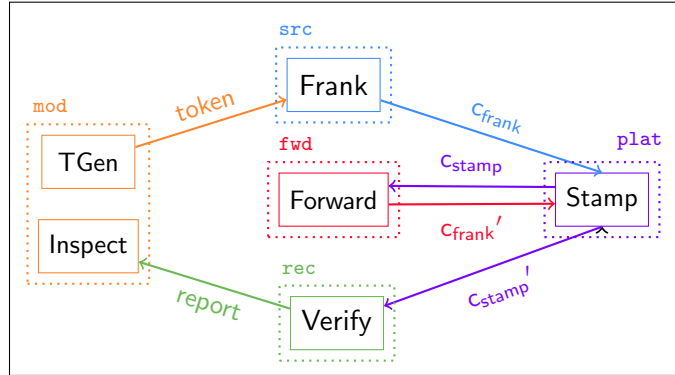


Figure 2.1: Diagram of Hecate’s data flow for a message m , from the source (top) to the forwarder (middle) and then to a reporting receiver (bottom). The commands match our definition of AMF with preprocessing (Def. 11), and the variables token , C_{frank} , C_{stamp} , and report are defined in Fig. 2.2.

Fig. 2.1. As before, the online work of message franking and transmission requires only one round of communication from the source to platform to receiver. Beforehand, we allow the source and moderator to engage in a single data-independent preprocessing interaction to produce *tokens* that can be consumed during the online phase. Preprocessing can be batched to produce many tokens at once, it can be performed during off-peak hours when the source’s device is connected to power and wifi, and it should be performed in advance rather than on the fly in order to avoid network-level traffic linking attacks [184]. As with MPC [42] or PIR [45], we show that adding a preprocessing round to AMF allows for more efficient protocols, and in particular allows us to answer the open question from above.

Concretely, we contribute an AMF scheme called Hecate. Our construction leverages the fact that, with preprocessing, the communication path of reported messages begins and ends with the content moderator. Ergo, we can use techniques from (faster) symmetric message franking whereby the moderator can prepare a token (e.g., a symmetric encryption of the source’s identity) that is only intelligible to its future self. The token is passed through the sender \rightarrow platform \rightarrow receiver communi-

cation flow of an EEMS; that said, end-to-end encryption prevents the platform from viewing the token.

Hecate also supports *source tracing*, in which receivers can forward messages along with their corresponding tokens. Any recipient can choose to report an abusive message; this only requires sending one communication to the moderator.

A big challenge in our construction is to combine message forwarding with our AMF *backward* (or post-compromise) security requirement, which states that an attacker who previously (but no longer) controlled a source’s device cannot blame the source for new messages. To our knowledge, this work is the first one to consider and formalize backward security within AMF. As we will discuss in more detail in §2.2, the challenge in combining AMF with backward security stems from the fact that indirect recipients of a message (after forwarding) cannot rely on the backward security of the underlying encrypted messaging protocol to tell whether the token is produced by the now-uncorrupted source or the attacker.

In summary, we make four contributions in this work.

- We rigorously define AMF with preprocessing (§2.3). We generalize the definition from TGLMR, formalize forward and backward security, and add source tracing.
- We provide a construction called **Hecate** (§2.5). It requires only a few black-box calls to standard crypto primitives.
- We formalize and prove (§2.6) that **Hecate** achieves all of the security guarantees shown in Table 2.1.
- We implement **Hecate** (§2.7) and integrate it into a Signal client. We show that **Hecate**’s performance compares favorably to prior work and is imperceptible in practice.

Before continuing, we wish to stress that any decision to use content moderation

within end-to-end encrypted messengers requires weighing all of its potential benefits and risks, including the limitations of *Hecate* and prior works (see §2.8.2), and the risk of abuse by (or coercion of) the moderator. This is a complex policy question whose discussion should involve computer scientists, but not only computer scientists. We take no stance on the policy question in this work; instead, we observe that these policy discussions are already ongoing [21, 66, 210] and that a sub-optimal understanding of the technological possibilities may push a service provider or nation-state policymakers toward a worse policy decision. We undertake this research in order to demonstrate the feasibility of alternatives to blunt privacy-inhibiting legislation.

2.2 Overview

In this section, we describe our objectives for an asymmetric message franking (AMF) system. We begin by describing the setting and threat model for our work, and then we provide a high-level description of the security requirements and a brief description of how our *Hecate* protocol will achieve them.

2.2.1 Setting and Threat Model

In this work, we consider an EEMS that might contain network-level anonymity protections such as Signal’s sealed sender [223] or Tor [101]. We focus on two party point-to-point communication; that said, our techniques translate directly to Signal’s group messaging protocol as described in §2.8.1.

Within the context of any single message transmission, we refer to the participating clients using the following terminology: the *source* who initially produced the message within the messaging platform, the *receivers* who receive the message and can optionally decide to report it (in which case we call them a *reporter*) and the *forwarders* who are receivers that decide to send the message along to others. All clients only possess the computational power of a phone.

Due to forwarding, the communication graph of each message has the structure of a tree rooted at the source. Several messages can be sent concurrently, and a client can have different roles in the communication trees of different messages.

In addition to the messenger clients, our model contains two (*possibly* separate, and more computationally powerful) entities that everyone can communicate with: (1) the *platform* that provides the messaging service, and (2) the content *moderator* that receives reports and helps victims of abuse. We consider the platform and moderator as possibly separate so that our model can capture settings where a platform outsources moderation tasks to other, more qualified organizations (e.g., Facebook’s oversight board [198]). We emphasize that our model and construction do not rely upon separation of these roles in any way; they remain valid even if the platform and moderator are operated by the same entity.

In general, the parties in the system view all other parties as potentially malicious and colluding together. Every party wants confidentiality and integrity to the strongest extent possible, even if some or all of their counterparty, the platform, and the moderator are colluding against them. In particular, we wish to retain all of the security goals that end-to-end encrypted messengers provide, as detailed in §2.2.2 and §2.6.

The parties’ relationships toward the moderator are subtle and merit further discussion. The moderator and platform view each other as semi-honest; looking ahead to our *Hecate* construction, the moderator trusts the accuracy of any timestamp applied by the platform but it need not trust the platform for any other purpose. Clients have a choice: if they view the moderator and platform as malicious and colluding, then they must be assured of limits to the moderator’s power; or, if they view the moderator as semi-honest then they must be assured that the moderator can perform its role.

In this work, a malicious attacker has the power to compromise one or more parties, in which case it can observe these parties' local state (e.g., cryptographic keys) and run arbitrary software for the duration of their control of a victim's machine. A semi-honest party, by contrast, is assumed to perform all actions honestly, and the only objective against such a party is data minimization. We presume that the software implementing the encrypted messenger faithfully reproduces the intended specification so that the adversary cannot control the behavior of honest parties. Put another way, supply chain attacks and formal verification are out of scope of this work.

The objective of holding senders accountable for reported messages creates a tension with the security goals of end-to-end encrypted messengers. In particular, clients no longer receive confidentiality, deniability, anonymity, or other privacy guarantees against the moderator for reported messages. Moreover, an AMF scheme imposes a limit on forward security, because messages sent in the past now can be revealed to the moderator in the present. Our objective is to ensure security up to these fundamental limits. We emphasize that even if the moderator is malicious and colluding with some clients, *all of the security guarantees for end-to-end encrypted messaging continue to hold for all unreported messages communicated between non-colluding clients*. Moreover, even for reported messages, security holds against all other parties who are not colluding with the moderator.

Another tension exists between content moderation and network anonymity. For example, *sealed sender* is a feature introduced by the Signal protocol to hide the identity of the sender from the platform. It offers sender confidentiality and minimizes the amount of metadata stored by the platform. But if the sender can deny ever sending a message, then can we hold anyone responsible for sending an abusive message? TGLMR [238] resolved this dilemma using zero-knowledge signatures. In

this paper, we contribute an alternative construction based solely on black-box use of standard crypto primitives.

2.2.2 Security goals

In an asymmetric message franking scheme, we aim to provide all of the security and privacy goals of encrypted messengers [85, 241]. Some EEMS goals (cf. §2.4.1) are already consistent with content moderation, in which case AMF constructions can use these properties and must ensure that they don't weaken them. To give a concrete example for our *Hecate* protocol: we use the EEMS as a black box, and we will take advantage of the receiver's ability to authenticate the sender's identity. On the other hand, some security goals are not fully compatible with content moderation, in which case we aim to make the smallest modification possible.

Below, we describe each security goal from Table 2.1 and highlight the extent to which it is impacted by content moderation. These security goals apply to all clients who construct properly formatted messages that adhere to the encrypted messaging protocol, whether or not their messages are subsequently reported. That is: even though malicious parties in a crypto protocol receive no security guarantees, the mere act of sending a reported message does not render a client malicious.

- *Confidentiality*. Anyone not involved in the creation, forwarding, or reporting of a message m must not learn anything about m except an upper bound on its size.
- *Anonymity*. The AMF scheme should not allow the platform, receiver, or moderator to learn anything about the source and forwarding path of a message beyond what they would learn from the underlying EEMS or a report.
- *Deniability*. Every user should be able to deny a claim about the contents of an unreported message made by any adversary (even other recipients of this

message). Also, reported messages are deniable to anyone other than the moderator, and if the moderator’s keys are breached then they become deniable to everyone.

- *Forward security.* Adversaries that compromise users’ state in the present should not be able to deduce anything about messages exchanged in the past. This goal does not apply to messages that happen to remain on the phone in the present, which can still be read and reported.
- *Backward security.* Once a client recovers from a compromise event, then the compromised state becomes ‘useless’ after a short delay. That is, the adversary cannot subsequently originate a new message that (if reported) would cause the moderator to blame the victim client.
- *Unforgeability.* The adversary cannot send a message that appears to be sent by another party. An honest receiver will reject any malformed or tampered messages.
- *Accountability.* If a message passes a receiver’s verification check and is subsequently reported, the moderator will trace it back to its original source. That is: nobody can falsely accuse someone who wasn’t the source of a message, and the true source cannot evade detection and yet also have the message verified by the receiver.

2.2.3 Protocol Overview

In this section, we give a high level overview of our **Hecate** protocol in two stages (with and without message forwarding) and explain informally how it satisfies our security goals.

Hecate without forwarding. At a high level, our **Hecate** construction can be thought of as an interactive variant of designated-verifier signatures. Given a message

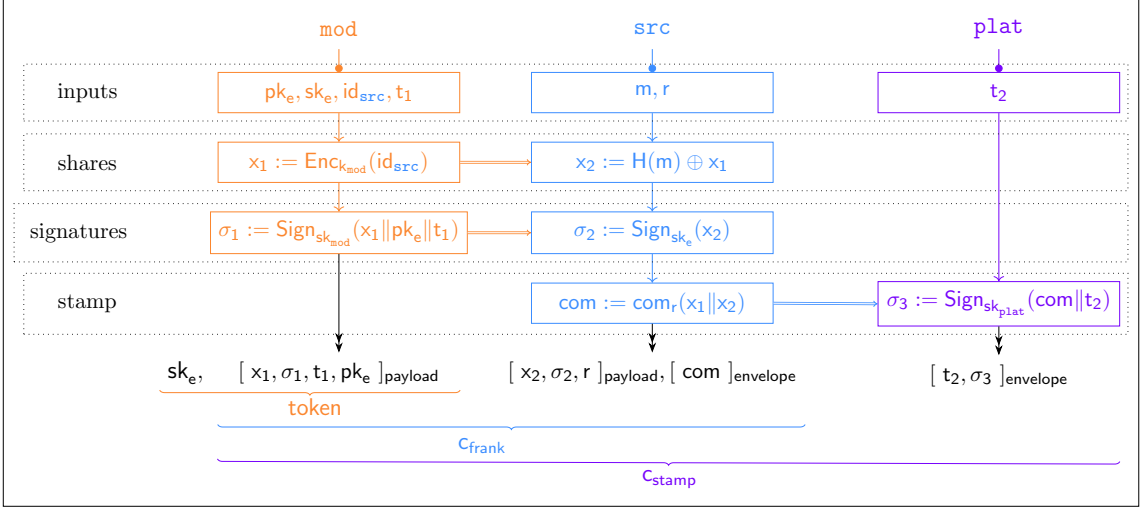


Figure 2:2: The construction of the different parts of a franked cipher. The outputs of the diagram correspond to each party’s contribution to the eventual stamped cipher c_{stamp} . The source constructs the franked cipher c_{frank} using a token provided by the moderator during preprocessing. We denote by **payload** and **envelope** two different parts of the ciphertext as defined in the Signal sealed sender protocol [223]; the platform and receiver can read the **envelope** whereas only the receiver can read the **payload**.

m , the source constructs a 2-out-of-2 secret sharing, say $H(m) = x_1 \oplus x_2$. In Hecate, the moderator binds x_1 to the source’s identity (which on its own reveals nothing about m), and then the source binds x_1 to x_2 without using any long-lived keys.

As shown in Fig. 2:2: since one of the two shares can be sampled even before m is known, during preprocessing the moderator selects x_1 as an encryption of the source’s identity (which appears random to everyone else), samples an ephemeral digital signature keypair (sk_e, pk_e) , and signs both x_1 and pk_e . The tuple of x_1 , pk_e , and their signature constitute the preprocessing token **token**. During the online phase, the source uses the ephemeral key sk_e to sign x_2 ; we refer to the pair of x_2 and its signature as another token **token**.

The source provides both tokens to the receiver within the payload of an ordinary Signal packet, as shown in Fig. 2:2; ignore the other elements of the franked ciphertext

c_{frank} for now. Any receiver can check on its own whether the signatures are valid and the underlying values x_1 and x_2 combine to form the real message m that the receiver also gets from the underlying Signal communication; if verification fails, then the message is malformed, so it is dropped without displaying on the receiver’s device. If a verified message is later reported, the two tokens together will convince the moderator that the source was the originator of message m .

Achieving our security goals. Many of our security guarantees follow directly from the corresponding property of the underlying EEMS, so we focus on the most challenging goals here. **Hecate** provides accountability for the same reason as symmetric messaging franking schemes: the moderator created an authenticated encryption of the source’s identity for its future self. Forward security holds because ephemeral signing keys sk_e from the past were deleted before a compromise event in the present. Deniability can be shown in two parts: if the moderator’s keys are breached then anyone can produce signatures for any choices of x_1 and x_2 , and otherwise the source’s identity is hidden within the encrypted token so anyone could have ‘forged’ signatures of an (x_1, x_2) pair using her own tokens rather than those of the real source.

Backward (or post-compromise [87]) security is more challenging to address, and it is worth pausing for a moment to discuss what this guarantee means in the context of content moderation. If an adversary corrupts the source’s phone, it *can* produce messages whose reports blame the source; this is inevitable. Our goal is to ensure that once the source recovers control of her phone, then (perhaps after a short delay δ) any new message produced by the adversary cannot implicate the honest source. To provide this guarantee within **Hecate**, the moderator includes a timestamp within its attestation to x_1 , and receivers drop any message where this timestamp is too old. This ensures that an adversary cannot use stale pre-processing tokens long after the compromise event.

Hecate with message forwarding. Next, we allow forwarding of messages and consider source tracing, in which the moderator should identify only the original source of a reported message. For the most part, our construction is already amenable to source tracing: a forwarder can simply include the original source’s tokens within a forwarded message rather than generating new tokens that would implicate herself. However, our timestamp-based solution to backward security now fails because the age of x_1 is insufficient to determine whether the original source had control of her cryptographic keys at the moment that the *original message* was sent (as opposed to the time of the forwarding).

We solve this problem by appending a timestamp **time** as the data traverses through the platform. To verify backward security, it suffices to verify whether the pre-processed token (which contains the identity of the source to blame) was produced close in time to the message transmission. Timestamps for forwarded messages are disregarded; future recipients only care about the timestamp from the original source.

It only remains to bind the source timestamp to the message, so that it cannot be tampered later. Note that we cannot reveal x_2 to the platform, or else the platform and moderator together could recover the content of messages. Blind signatures are a possible solution to allow the platform to timestamp-and-sign obliviously, but constructions that only require one message received and sent by the platform require trusted setup [114], non-standard crypto assumptions [120, 121], or a concretely slow runtime with non-black-box reductions [115, 157]. Instead, we take advantage of the fact that the platform’s actions need only be verified by recipients who already know x_1 and x_2 , so it suffices for the platform to produce a signature σ of the current **time** together with a commitment to the two shares. The corresponding decommitment randomness can be sent to the receiver within the encrypted messenger payload, so that the recipient can verify that it is well-formed.

2.3 Definitions

In this section, we present a nearly black-box model of the Signal protocol that we use in this work, and then we detail a new definition for an asymmetric message franking scheme that generalizes TGLMR [238].

2.4 Definitions of Cryptographic Building Blocks

This work uses four standard cryptographic building blocks that we use and adapt from Boneh-Shoup [59] and Katz-Lindell [158]. In what follows, we define the message space as $\mathcal{M} := \{0, 1\}^*$, the key space as $\mathcal{K} := \{0, 1\}^n$, the ciphertext space as $\mathcal{C} := \{0, 1\}^*$, the randomness space $\mathcal{R} := \{0, 1\}^n$ and the signature space as $\Sigma := \{0, 1\}^n$, where n denotes the security parameter.

Definition 1 (Commitment scheme). *A non-interactive commitment scheme is defined by two algorithms Com and Vf .*

- Com is an algorithm that takes a random string $r \leftarrow \mathcal{R}$, and a plaintext message $m \in \mathcal{M}$ and outputs a commitment $\text{com} := \text{Com}(m, r)$.
- Vf is an algorithm that takes a commitment com , a string r and a plaintext message m and checks if $\text{Vf}(m, \text{com}, r) := (\text{Com}(m, r) \stackrel{?}{=} \text{com})$.

Definition 2 (Binding commitment). *A commitment scheme $\pi = \{\text{Com}, \text{Vf}\}$ is computationally binding if for all probabilistic polynomial time (PPT) adversaries \mathcal{A} , there is a negligible function $\text{negl}(n)$ such that:*

$$\text{Adv}_{\pi}^{\text{binding}_{\text{com}}}(\mathcal{A}) = \Pr[\text{Com}(m, r, \text{param}) = \text{Com}(m', r', \text{param}) \mid m \neq m'] \leq \text{negl}(n).$$

Definition 3 (Hiding commitment). *Let $\pi = \{\text{Com}, \text{Vf}\}$ be a commitment scheme.*

Let $\text{Com}_{\text{hiding}}^{\mathcal{A}}$ be defined by the following experiment:

- The adversary \mathcal{A} outputs a pair of messages $m_0, m_1 \in \mathcal{M}$.
- A uniform bit $b \in \{0, 1\}$ and the randomness $r \leftarrow \{0, 1\}^n$ are chosen.
- The adversary \mathcal{A} is given access to the commitment oracle $\text{O}^{\text{com-hiding}}$ which on messages m_0 and m_1 computes and returns the commitment $\text{com} \leftarrow \text{Com}(m_b, r)$, where $\text{Vf}(\text{Com}(m_b, r), m_b, r) = 1$.
- The output of the experiment is 1 if $b' = b$ and 0 otherwise.

A commitment scheme π is computationally hiding if for all PPT adversaries \mathcal{A} there is a negligible function $\text{negl}(n)$ such that:

$$\text{Adv}_{\pi}^{\text{hiding-com}}(\mathcal{A}) = \Pr[\text{Com}_{\text{hiding}}^{\mathcal{A}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n).$$

Definition 4 (Encryption scheme). A randomized private key encryption scheme is defined by three algorithms EncKGen , Enc and Dec over a finite message space \mathcal{M} .

- EncKGen is a probabilistic key generation algorithm that outputs a key sk sampled uniformly at random from \mathcal{K} .
- Enc is the randomized encryption algorithm that takes as an input sk and plaintext message $m \in \mathcal{M}$ and outputs $c := \text{Enc}_{\text{sk}}(m)$ where $c \in \mathcal{C}$.
- Dec is the decryption algorithm that takes as an input sk and a ciphertext c in the ciphertext space \mathcal{C} and outputs a plaintext message $m := \text{Dec}_{\text{sk}}(c)$ such that $c := \text{Enc}_{\text{sk}}(m)$.

Definition 5 (CCA security). Let $\pi = \{\text{EncKGen}, \text{Enc}, \text{Dec}\}$ be an encryption scheme.

Let $\text{ENC}_{\text{cca}, \pi}^{\mathcal{A}}(n)$ denote the following experiment:

- EncKGen is run to obtain (pk, sk) and a uniform bit $b \in \{0, 1\}$ is chosen. The adversary \mathcal{A} is given pk .

- The adversary \mathcal{A} is given access to the encryption oracle $\mathcal{O}_{cca}^{\text{enc}}$ which, on messages $\mathbf{m}_0, \mathbf{m}_1$, outputs a ciphertext $c \leftarrow \text{Enc}_{\text{pk}}(\mathbf{m}_b)$.
- The adversary \mathcal{A} is given access to the decryption oracle $\mathcal{O}_{cca}^{\text{decrypt}}$ which outputs the decrypted plaintext message \mathbf{m} under sk when handed out a ciphertext c' .
- \mathcal{A} continues to interact with the decryption and encryption oracles, but may not request a decryption of any ciphertext c returned by $\mathcal{O}_{cca}^{\text{enc}}$.
- Finally \mathcal{A} output a bit b' . The output of the experiment is defined to be 1 if $b = b'$, and 0 otherwise.

We say that π is secure under a chosen-ciphertext attack (CCA) if for all PPT adversaries \mathcal{A} , there exists a negligible function $\text{negl}(n)$ such that:

$$\text{Adv}_{\pi}^{\text{enc}_{cca}} = \Pr[\text{ENC}_{cca,\pi}^{\mathcal{A}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n).$$

Definition 6 (CPA security). Let $\pi = \{\text{EncKGen}, \text{Enc}, \text{Dec}\}$ be an encryption scheme. Let $\text{ENC}_{cpa,\pi}^{\mathcal{A}}(n)$ denote a similar experiment to $\text{ENC}_{cca,\pi}^{\mathcal{A}}(n)$ where the adversary \mathcal{A} only has access to the encryption oracle that rename as $\mathcal{O}_{cpa}^{\text{enc}}$. We say that π is secure under a chosen-plaintext attack (CPA) if for all PPT adversaries \mathcal{A} , there exists a negligible function $\text{negl}(n)$ such that:

$$\text{Adv}_{\pi}^{\text{enc}_{cpa}} = \Pr[\text{ENC}_{cpa,\pi}^{\mathcal{A}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n).$$

Definition 7 (Digital signature scheme). A signature scheme is defined as the triple of algorithms $\text{SigKGen}, \text{Sign}, \text{Vf}$ over the message space \mathcal{M} and the signature space Σ .

- SigKGen is a probabilistic key generation algorithm that output a key pair (pk, sk) sampled from \mathcal{K} , where pk is the public verification key and sk is the secret

signing key.

- **Sign** is the probabilistic signing algorithm that takes the signing key \mathbf{sk} and a plaintext message \mathbf{m} and outputs a signature $\sigma \leftarrow \text{Sign}_{\mathbf{sk}}(\mathbf{m})$, where $\sigma \in \Sigma$.
- **Vf** is the deterministic verification algorithm which checks the signature σ against the plaintext message \mathbf{m} and public key \mathbf{pk} and outputs \perp or 1 such that:

$$\Pr[\text{Vf}(\mathbf{pk}, \mathbf{m}, \text{Sign}_{\mathbf{sk}}(\mathbf{m})) = 1] = 1.$$

Definition 8 (EU-CMA security). Let $\pi = \{\text{SigKGen}, \text{Sign}, \text{Vf}\}$ denote a digital signature scheme. Let $\text{Sig}_{\text{eu-cma}}^{\mathcal{A}}$ be the experiment defined as:

- **SigKGen** is run to obtain $(\mathbf{pk}, \mathbf{sk})$.
- The adversary \mathcal{A} is given \mathbf{pk} and access to the signing oracle $\mathcal{O}_{\text{eu-cma}}^{\text{sign}}$ which on message \mathbf{m} computes and outputs the signature σ of that message under the secret signing key \mathbf{sk} . Let \mathcal{Q} denote the set of all queries that \mathcal{A} makes to $\mathcal{O}_{\text{eu-cma}}^{\text{sign}}$.
- The adversary \mathcal{A} then outputs (\mathbf{m}', σ') .
- The experiment outputs 1 if and only if $\text{Vf}_{\mathbf{pk}}(\mathbf{m}', \sigma') = 1$ and $\mathbf{m}' \notin \mathcal{Q}$, and 0 otherwise.

We say that π is existentially unforgeable under an adaptive chosen-message attack (EU-CMA) if for all PPT adversaries \mathcal{A} , there is a negligible function $\text{negl}(n)$ such that:

$$\text{Adv}_{\pi}^{\text{sig}_{\text{eu-cma}}}(\mathcal{A}) = \Pr[\text{Sig}_{\text{eu-cma}}^{\mathcal{A}}(n) = 1] \leq \text{negl}(n).$$

Definition 9 (Hash function). A hash function with output length l is defined by two algorithms **Gen** and **H**.

- **Gen** is a probabilistic algorithm which outputs a key $k \in \mathcal{K}$.

- H is an algorithm which takes as input as key k and a string $m \in \mathcal{M}$ and outputs a string $H_k(m) \in \{0, 1\}^{l(n)}$.

Definition 10 (Collision resistance). Let $\pi = \{\text{Gen}, H\}$ denote a hash function. Let $\text{Hash}_{\text{coll}}^A$ be the experiment defined as:

- Gen is run to obtain k .
- The adversary \mathcal{A} is given access to the hashing oracle $\mathcal{O}^{\text{hash}}$ which on input m returns $H_k(m)$
- The adversary then outputs m_0 and m_1 .
- The experiment outputs 1 if and only if $m_0 \neq m_1$ and $H_k(m_0) = H_k(m_1)$.

We say that π is collision resistant if for all PPT adversaries \mathcal{A} there is a negligible function $\text{negl}(n)$ such that:

$$\text{Adv}_{\pi}^{\text{hash}_{\text{coll}}}(\mathcal{A}) = \Pr[\text{Hash}_{\text{coll}}^A(n) = 1] \leq \text{negl}(n).$$

2.4.1 Modeling an EEMS

End-to-end encrypted messaging systems (EEMS) using the Signal protocol [225] are a complex, delicate combination of standard cryptographic primitives. Starting with Cohn-Gordon et al. [85], there is a long line of research that analyzes the security of EEMS constructions such as the two-party Signal protocol itself (e.g., [27, 47, 151]), modified versions that provide provide stronger guarantees (e.g., [148, 150, 208]), and extensions to support group messaging (e.g., [69, 77, 214]).

In this work, we wish to treat an EEMS as a black box and consider only an API-level description of its operation and underlying security guarantees. For this reason, we follow the UC models of an EEMS as shown by the recent works of Bienstock et al. [53] and Canetti et al. [71], rather than the game-based definitions in the

literature (e.g., [27, 47]). The ideal functionality $\mathcal{F}_{\text{Signal}}$ in these works models the creation, evolution, and destruction of communication ‘sessions’ between different pairs of parties, and keeps track of the long-term and ephemeral state that parties must hold for each operation.

We follow this abstract model, with three changes. First, we allow a sender to attach public information outside of the encrypted message on the envelope that is visible to the platform, as shown in Fig. 2.2. As defined by Signal’s sealed sender construction [223], the *envelope* constitutes the outer shell of an encrypted message and contains any data that is visible to the platform such as the ciphertext and the recipient’s address. Second, in order to support an anonymous network, we allow for inputs involving the parties’ identities to be optional. Third, we add an explicit forgery method to highlight the fact that an EEMS achieves deniable authentication [103] (which is implicitly true in universally composable security models [73]): that is, the sender and receiver can forge a transcript showing that a message originated with the other party.

Hence, our abstract model of Signal involves three methods. All of these methods implicitly use the state of the party (or parties) that participate in each method.

- $\text{send}_{\text{eems}}(m^*; \text{id}_{\text{src}}, \text{id}_{\text{rec}}) \rightarrow c$: Run by the source client id_{src} with message m^* , this method sends a ciphertext c to the platform. This message m^* might contain payload and envelope components, similarly to how Signal’s sealed sender operates. We sometimes omit the latter two inputs when they are clear from context.
- $\text{deliver}_{\text{eems}}(c; \text{id}_{\text{rec}}) \rightarrow m^*$: An interactive protocol in which the platform delivers a ciphertext c to the receiver id_{rec} . If this receiver was the intended target of a previous $\text{send}_{\text{eems}}$ that produced c , then they can decrypt using their local state to recover m^* . As above, we presume that $\text{deliver}_{\text{eems}}$ handles the payload and

envelope of the message and splits c and m^* accordingly. Here, it is unclear whether to include id_{rec} as an input: for an anonymous communication channel it is important that the platform not know id_{rec} , but for non-anonymous networks it may be required. We leave id_{rec} as an optional parameter, and throughout this work we focus on the stronger setting in which the network is anonymous so this input is omitted.

- $\text{forge}_{\text{eems}}(m^*; \text{id}_{\text{src}}, \text{id}_{\text{rec}}) \rightarrow c$: A forgery algorithm executed by a party id_{rec} and requiring its state $\text{state}_{\text{rec}}$. It forges a transcript that looks as though the message m^* were sent by its counterparty id_{src} in an EEMS communication, with a destination of id_{rec} . The parameters id_{src} and id_{rec} are optional for the same reasons as $\text{send}_{\text{eems}}$, and they will be omitted from this work.

2.4.2 Defining AMF with Preprocessing

Next, we present a rigorous definition for an asymmetric message franking system with preprocessing. This definition extends the one from TGLMR [238] in two ways. First, it includes an (optional) out-of-band communication between the moderator and sender, which results in a one-time *token* that is consumed when sending a message. Second, it is designed in a modular fashion so that it can be built on top of any EEMS that adheres to the model in §2.4.1.

Definition 11. *An asymmetric message franking scheme with preprocessing $\text{AMF} = (\text{KGen}, \text{TGen}, \text{Frank}, \text{Forward}, \text{Stamp}, \text{Inspect}, \text{Verify}, \text{Forge}_{\text{mod}}, \text{Forge}_{\text{rec}})$ is a tuple of algorithms called by different parties in the messaging ecosystem. We assume that each party has a unique identifier id provided by the underlying EEMS, and we define a state variable state for each party containing all keys and tokens generated by the AMF scheme and the underlying EEMS that have not yet been deleted. (Note that the user’s state does not contain a transcript of prior messages exchanged.) The*

algorithms operate as follows.

- $(pk, sk) \leftarrow \text{KGen}()$: The key generation algorithm accessed by any party in the EEMS and used for creating (potentially multiple) cryptographic keys. The algorithm is at least run once at the beginning of time to setup the long-term key material for each party.
- $\text{TGen}(id_{\text{src}}, time_{\text{mod}}, k_{\text{mod}}) \rightarrow \text{token}$: An algorithm run by the moderator periodically that provides a one-time token for use when sending a message. An honest moderator should only provide tokens to a participant that correspond to their actual identity id_{src} . It is assumed that the moderator can rely on the EEMS to authenticate a user's identity id_{src} before running TGen.
- $\text{Frank}(state_{\text{src}}, m, id_{\text{rec}}, \text{token}) \rightarrow m_{\text{frank}}$: The message franking algorithm that allows a user with state $state_{\text{src}}$ to frank a plaintext message m that they wish to send to a receiver id_{rec} , using the token received during preprocessing. The $state_{\text{src}}$ contains all key material produced by KGen and the underlying EEMS, although Frank need not use this state. The resulting franked message m_{frank} can be sent to the platform using $\text{send}_{\text{eems}}$.
- $\text{Stamp}(c_{\text{frank}}, sk_{\text{plat}}, \text{time}) \rightarrow c_{\text{stamp}}$: The stamping procedure run by the platform to authenticate and timestamp a franked cipher c_{frank} . The resulting stamped cipher c_{stamp} can then be delivered to its intended recipient using the $\text{deliver}_{\text{eems}}$ method. Stamp does not have the sender or receiver's identity, even if $\text{deliver}_{\text{eems}}$ does.
- $\text{Forward}(m_{\text{frank}}, state_{\text{fwd}}, id_{\text{rec}}) \rightarrow m_{\text{frank}}'$: Forwarding algorithm that allows a user with franked message m_{frank} to produce a new franked message m_{frank}' intended for a new recipient id_{rec} . The format of m_{frank} and m_{frank}' are identical, so the ciphertexts of new and forwarded messages look indistinguishable to the plat-

form.

- $\text{Verify}(\mathbf{m}_{\text{frank}}, \text{state}_{\text{rec}}) \rightarrow (\mathbf{m}, \text{report})$ or \perp : The report construction algorithm that allows a receiver to validate a franked message $\mathbf{m}_{\text{frank}}$ with respect to its state $\text{state}_{\text{rec}}$. If valid, Verify returns the corresponding plaintext message \mathbf{m} along with a string report that the receiver can send to the moderator if they choose to report an abusive message.
- $\text{Inspect}(\text{report}, \mathbf{k}_{\text{mod}}) \rightarrow (\text{id}_{\text{src}}, \mathbf{m}, \text{time})$ or \perp : The inspection algorithm that allows a moderator to handle reported message report using their secret key \mathbf{k}_{mod} by validating and possibly source tracing them. If the verification step succeeds, the moderator produces the id of the source id_{src} , the message contents \mathbf{m} , and a timestamp of the message time .
- $\text{Forge}_{\text{mod}}(\text{id}_{\text{src}}, \text{id}_{\text{rec}}, \mathbf{m}, \mathbf{k}_{\text{mod}}) \rightarrow \mathbf{m}_{\text{frank}}$: For deniability, this forgery protocol allows a moderator with secret key \mathbf{k}_{mod} to forge a franked message with plaintext \mathbf{m} on behalf of a user with id id_{src} and with an intended recipient with id id_{rec} .
- $\text{Forge}_{\text{rec}}(\text{id}_{\text{rec}}, \mathbf{m}, \text{state}_{\text{rec}}; \text{id}_{\text{src}}) \rightarrow \mathbf{c}_{\text{frank}}$: For deniability, this forgery algorithm allows a receiver with id id_{rec} and state $\text{state}_{\text{rec}}$ to forge a franked ciphertext as though the message \mathbf{m} was transmitted through the EEMS by the sender id_{src} to the receiver id_{rec} . Note that id_{src} is an optional parameter and may not be needed by systems that support anonymous messaging. In this work, we omit it from the presentation of this work since we are aiming for the highest level of anonymity.

We say that an AMF scheme with preprocessing is secure if all computationally bounded attackers have negligible advantage in winning the deniability, anonymity, confidentiality, accountability, and backward secrecy games. These games are nuanced to describe, so rather than doing so here, we defer our discussion to the security analysis in §2.6.

Command	Actor	KeyGen	Sign	Verify
TGen	mod	1	1	0
Frank	src	0	1	0
Stamp	plat	0	1	0
Forward	fwd	0	0	0
Verify	rec	0	0	3
Inspect	mod	0	0	3

Table 2.2: The number of public-key digital signature operations required for each of the interactive algorithms within **Hecate** (except for the one-time KGen at setup). We only count the additional cryptographic operations required for **Hecate** beyond those already required by the EEMS.

2.5 Constructing Hecate

In this section, we describe the **Hecate** construction in detail. As per Def. 11, **Hecate** has eight algorithms. We describe them within this section, and we provide the full protocol specification of **Hecate** in Figs. 2-3-2-4. Because they are the most expensive of our standard crypto primitives, we also count the number of public key operations in each step here and in Table 2.2. For context, the prior AMF scheme from TGLMR [238] required at least 11 modular exponentiations per algorithm.

Key generation. KGen initializes a few long-term keys: the moderator samples an authenticated encryption key k_{mod} and both the moderator and platform sample a digital signature key pair $(pk_{\text{mod}}, sk_{\text{mod}})$ and $(pk_{\text{plat}}, sk_{\text{plat}})$. One strength of **Hecate** is that individual parties do not need any long-term key material besides their existing EEMS keys, which simplifies our analysis of forward and backward security.

Token generation during preprocessing. In TGen, the moderator creates a batch of tokens for users at specific time intervals. Each token provides users with:

- Ephemeral session keys (pk_e, sk_e) that they can use to sign their message. None of the keys tie to users' long-term key material, thus giving the sender plausible deniability and confidentiality with respect to other users.
- A dual purpose randomized encryption $x_1 := \text{Enc}_{k_{\text{mod}}}(\text{id}_{\text{src}})$ of the user's identity

id_{src} under the moderator’s secret key k_{mod} that enforces accountability with respect to the moderator, confidentiality with respect to other users, and provides token integrity. The latter property is ensured by having the sender create a share x_2 that along with x_1 reconstructs to a hash of the sent message.

- A timestamp t_1 that provides backward security.
- A signature σ_1 by the moderator of the entire token that guarantees integrity and unforgeability of the token.

As shown in Table 2.2, the moderator requires two public key operations for token generation: 1 keygen operation to produce the ephemeral key pair and 1 signature to sign the public ephemeral key with the identity of the sender.

Message franking. The **Frank** method is executed every time the source wishes to send a message. The `constructfrank` procedure requires an input plaintext message m from the source and consumes a single token at a time, and it produces a franked message m_{frank} . This can be combined with `sendeems` to relay a franked ciphertext c_{frank} to the platform.

To produce the franked message, the sender begins by unpacking x_1 from the token and computes x_2 such that these variables constitute a 2-out-of-2 sharing of $H(m)$. Next, x_2 is signed via the ephemeral keys in the original token to produce σ_2 . Collectively, x_2 , σ_2 , and elements of the pre-processing token (excluding the secret ephemeral key) will constitute the payload of the franked message. Then, the sender creates a commitment `com` of $x_1 || x_2$ using the randomness r . The user then pushes `com` onto the envelope of the franked message and appends r to its payload. In total, a sender only requires 1 public key operation to sign the second share x_2 . The constructed franked message m_{frank} has several properties: x_2 and `com` bind the online and preprocessing stages together, the signature allows the receiver to check the well-formedness of the message, and the use of an ephemeral signing key provides

KGen [**mod** and **plat**, separately]

```

1 :  $k_{\text{mod}} \leftarrow \$ \text{EncKGen}(1^n)$ 
2 :  $(pk_{\text{mod}}, sk_{\text{mod}}) \leftarrow \$ \text{SigKGen}(1^n)$ 
3 :  $(pk_{\text{plat}}, sk_{\text{plat}}) \leftarrow \$ \text{SigKGen}(1^n)$ 

```

TGen [**mod** \rightarrow **src**]

```

1 :  $\text{token} := \text{construct}_{\text{token}}(sk_{\text{mod}}, id_{\text{src}})$ 
2 : return token

```

Frank [**src** \rightarrow **plat**]

```

1 :  $m_{\text{frank}} := \text{construct}_{\text{frank}}(\text{token}, m)$ 
2 :  $c_{\text{frank}} := \text{send}_{\text{eems}}(m_{\text{frank}})$ 
3 : return  $c_{\text{frank}}$ 

```

Forward [**fwd** \rightarrow **rec**]

```

1 :  $m_{\text{frank}}' := \text{construct}_{\text{fwd}}(m_{\text{frank}})$ 
2 :  $c_{\text{frank}} := \text{send}_{\text{eems}}(m_{\text{frank}}')$ 
3 : return  $c_{\text{frank}}$ 

```

Stamp [**plat** \rightarrow **rec**]

```

1 :  $c_{\text{stamp}} := \text{stamp}_{\text{time}}(c_{\text{frank}})$ 
2 :  $c_{\text{stamp}}' := \text{send}_{\text{eems}}(c_{\text{stamp}})$ 
3 : return  $c_{\text{stamp}}'$ 

```

Verify [**rec** \rightarrow **mod/rec**]

```

1 :  $m_{\text{frank}} := \text{deliver}_{\text{eems}}(c_{\text{stamp}})$ 
2 :  $m_{\text{frank}} := \text{move}_{\text{stamp}}(m_{\text{frank}})$ 
3 :  $(m, \text{report}) := \text{vfRec}(m_{\text{frank}})$ 
4 : if  $(m, \text{report}) \stackrel{?}{=} \perp$  :
5 :   return  $\perp$ 
6 : return  $(m, \text{report})$ 

```

Inspect [**mod**]

```

1 : if  $\text{vfMsg}(\text{report})$  :
2 :   return  $(\text{Dec}(\text{report}.x_1), \text{report}.t_2)$ 
3 : return  $\perp$ 

```

Figure 2-3: Hecate’s construction along with the transmissions using the encrypted messenger. The notation $[a \rightarrow b]$ means that party a executes the method and sends the returned value to b . Note that the **plat** relays messages between the **src** and the **rec**. The receiver of a message may elect to forward or report it; we assume that **Forward** is preceded by a successful invocation of **Verify**. See Fig. 2-4 for more details on the methods used here.

deniability with respect to anyone other than the moderator.

Stamping. In **Stamp**, the platform timestamps and digitally signs the envelope of a franked cipher c_{frank} ; ergo, this procedure requires 1 public key operation. Then, the platform relays the resulting stamped cipher c_{stamp} to its intended recipient. Stamping prevents a preprocessing token from being used indefinitely after a compromise to blame a victim client for unsent messages.

Verification and reporting. Upon reception, the receiver executes **Verify** to validate the signatures, timestamps, packet integrity, and envelope commitments. If a message

```

constructtoken( $k_{\text{mod}}$ ,  $sk_{\text{mod}}$ ,  $id_{\text{src}}$ )


---


1:  $(pk_e, sk_e) \leftarrow \$ \text{SigKGen}(1^n)$ 
2:  $t_1 := \text{time}()$ 
3:  $x_1 := \text{Enc}_{k_{\text{mod}}}(\text{id}_{\text{src}})$ 
4:  $\sigma_1 := \text{Sign}_{sk_{\text{mod}}}(x_1 \| pk_e \| t_1)$ 
5:  $\text{token} := (x_1, t_1, \sigma_1, (pk_e, sk_e))$ 
6: return token

constructfrank( $m$ , token)


---


1:  $r \leftarrow \$ \{0, 1\}^n$ 
2:  $(x_1, t_1, \sigma_1, (pk_e, sk_e)) = \text{token}$ 
3:  $x_2 := \text{split}(x_1, H(m))$ 
4:  $\sigma_2 := \text{Sign}_{sk_e}(x_2)$ 
5:  $\text{com} := \text{com}_r(x_1 \| x_2)$ 
6:  $\text{envelope} := \text{com}$ 
7:  $\text{payload} := (x_1, x_2, pk_e, r, t_1, \sigma_1, \sigma_2)$ 
8:  $\text{fwdpayload} := \emptyset$  // reserved for forwarder
9:  $m_{\text{frank}} := (m, \text{payload}, \text{fwdpayload}, \text{envelope})$ 
10: return  $m_{\text{frank}}$ 

constructfwd( $m_{\text{frank}}$ )


---


1:  $m_{\text{frank}} := \text{move}_{\text{stamp}}(m_{\text{frank}})$ 
2:  $m_{\text{frank}}.\text{envelope} \leftarrow \$ \{0, 1\}^n$ 
3: return  $m_{\text{frank}}$ 

stamptime( $c_{\text{frank}}$ ,  $sk_{\text{plat}}$ )


---


1:  $t_2 = \text{time}()$ 
2:  $\sigma_3 := \text{Sign}_{sk_{\text{plat}}}(\text{com} \| t_2)$ 
3:  $c_{\text{stamp}}.\text{envelope} := (\text{com} \| \sigma_3 \| t_2)$ 
4:  $c_{\text{stamp}}.\text{payload} := c_{\text{frank}}.\text{payload}$ 
5: return  $c_{\text{stamp}}$ 

movestamp( $m_{\text{frank}}$ )


---


1: if  $m_{\text{frank}}.\text{fwdpayload} \stackrel{?}{=} \emptyset$  :
2:    $m_{\text{frank}}.\text{fwdpayload} := m_{\text{frank}}.\text{envelope}$ 
3: return  $m_{\text{frank}}$ 

vfRec( $m_{\text{frank}}$ )


---


1: if  $\text{vfMsg}(m_{\text{frank}})$  :
2:    $\text{report} := m_{\text{frank}}$ 
3:   return  $(m_{\text{frank}}.m, \text{report})$ 
4: return  $\perp$ 

vfMsg(report)


---


1:  $b_1 := \text{vfToken}(\text{report})$ 
2:  $b_2 := \text{vfCom}(\text{report})$ 
3:  $b_3 := \text{vfExp}(\text{report})$ 
4: return  $b_1 \wedge b_2 \wedge b_3$ 

vfToken(report)


---


1:  $\text{reveal} := \text{open}(x_1, x_2)$ 
2:  $b_1 := (\text{reveal} \stackrel{?}{=} H(m))$ 
3:  $b_2 := \text{Vf}_{pk_{\text{mod}}}(x_1 \| pk_e \| t_1, \sigma_1)$ 
4:  $b_3 := \text{Vf}_{pk_e}(x_2, \sigma_2)$ 
5: return  $b_1 \wedge b_2 \wedge b_3$ 

vfExp(report)


---


1:  $b := |t_1 - t_2| \stackrel{?}{<} \text{expiry}$ 
2: return  $b$ 

vfCom(report)


---


1:  $(\text{com}, \sigma_3, t_2) \leftarrow \text{report}.\text{fwdpayload}$ 
2:  $b_1 := \text{Vf}(x_1 \| x_2, \text{com}, r)$ 
3:  $b_2 := \text{Vf}_{pk_{\text{plat}}}(\text{com} \| t_2, \sigma_3)$ 
4: return  $b_1 \wedge b_2$ 

```

Figure 2-4: Hecate’s subroutines. We omit writing out attribute access notation when it is obvious from the context (i.e. com for instance is a shorthand for $c_{\text{frank}}.\text{com}$).

fails the verification check, then the receiver drops the packet and the application never displays the plaintext message. Otherwise, `Verify` generates a plaintext message m that can be displayed on the receiver’s phone, and a `report` that can be sent out to the moderator if the receiver so chooses.

In `Hecate`, the `report` solely consists of the franked message m_{frank} . When the moderator receives a `report`, they locally run the `Inspect` method which performs the same verification procedure as the recipient, and if successful, decrypts the source’s identity from the ciphertext x_1 within the token. Both the receiver of a message and a moderator who receives a `report` require 3 signature verifications to check that the two shares are not tampered with and have the right timestamps.

Message forwarding. Verified messages can alternatively be forwarded using the optional `Forward` method. There are two differences between `Forward` and `Frank`: the forwarder creates a dummy commitment outside the Signal envelope, and it moves the true commitment and signed timestamp into the payload of the franked message. Because it reuses the prior signature, the forwarder doesn’t perform any public key operations of its own. Note that `Frank` and `Forward` payloads are distinguishable by the receiver but indistinguishable to the platform (meaning that it will stamp a forwarded cipher); see Table 2.3 for the format of m_{frank} . The receiver of a forwarded message executes `Verify` using the commitment, signature, and timestamp inside the payload (ignoring the envelope).

We defer discussion of `Hecate`’s forgery algorithms to §2.6, since these are proof artifacts of the deniability property rather than actual elements of the construction.

2.6 Security Analysis

In this section, we formally define the security properties of asymmetric message franking (AMF) with preprocessing, and we prove that `Hecate` guarantees them. All

<pre> send_{amf}(m, id_{src}, id_{rec}) ----- 1 : state_{src} := retrieve_{state}(id_{src}) 2 : fetch token from state_{src} 3 : m_{frank} := 4 : Frank(state_{src}, m, id_{rec}, token) 5 : C_{frank} := send_{eems}(m_{frank}) 6 : return C_{frank} receive_{amf}(C_{frank}, id_{rec}, time_{plat}, sk_{plat}) ----- 1 : C_{stamp} := Stamp(C_{frank}, id_{rec}, time_{plat}, sk_{plat}) 2 : m_{frank} := deliver_{eems}(C_{stamp}) 3 : state_{rec} := retrieve_{state}(id_{rec}) 4 : if Verify(m_{frank}, state_{rec})[?] = 0 : 5 : return ⊥ 6 : return m_{frank} fwd_{amf}(m_{frank}, id_{fwd}, id_{rec}) ----- 1 : state_{fwd} := retrieve_{state}(id_{fwd}) 2 : m_{frank}' := 3 : Forward(state_{fwd}, m_{frank}, id_{rec}) 4 : C_{frank} := send_{eems}(m_{frank}') 5 : return C_{frank} </pre>	<pre> O^{corrupt}(id) ----- 1 : // global_t is only relevant in BAC 2 : corrupted = corrupted ∪ (id, global_t) 3 : state_{id} := retrieve_{state}(id) 4 : return state_{id} O^{request}(id) ----- 1 : // global_t is only relevant in BAC 2 : if (id, global_t) ∈ corrupted : 3 : create a batch of d tokens 4 : using TGen() for id 5 : T := T ∪ token 6 : global_t := global_t + 1 7 : return d tokens 8 : return ⊥ </pre>
---	---

Figure 2.5: Game subroutines and oracles used in several games. Here, d is a fixed parameter known to the moderator.

of our definitions are written as indistinguishability games $\text{GAME}_b^{\mathcal{A}}$ for $b \in \{0, 1\}$, and we want to show that the adversary's advantage

$$\text{Adv}_{\text{Hecate}}^{\text{game}}(\mathcal{A}) = |\Pr[\text{GAME}_1^{\mathcal{A}} = 1] - \Pr[\text{GAME}_0^{\mathcal{A}} = 1]|$$

is negligible for each game, if the adversary \mathcal{A} is computationally bounded to probabilistically polynomial time (PPT).

2.6.1 Deniability

Deniability states that *a sender should always be able to deny that they sent a particular message to anyone, except to the moderator when a message is reported*. Deniability could hold with respect to a colluding moderator and receivers, as shown in the DENM_b^A game in Fig. 2-6, or against malicious receivers who are not colluding with an honest moderator, which corresponds to the DENR_b^A game in Fig. 2-6.

Each deniability game provides the adversary \mathcal{A} with polynomially-many queries to an oracle O_b^{DENM} or O_b^{DENR} , respectively. For each query, the adversary chooses a plaintext message m and the sender id_{src} and corrupted receiver $\text{id}_{\text{rec},\mathcal{A}}$ of that message. Both oracles behave similarly: depending on the parameterized choice bit b , the oracle will either forge a message as a corrupted moderator/receiver as if originating from id_{src} , or ask the honest source id_{src} to produce it themselves. Deniability requires that no adversary can distinguish between forged and real messages, even with access to the secret keys of malicious parties. In other words, we want to show that a user can always repudiate having sent a message even when the moderator/receiver provides access to their secret key material. The knowledge of the original source of a message is non-transferable in that sense. Additionally, O_b^{DENR} provides an interesting guarantee: since the receiver forgery $\text{Forge}_{\text{rec}}$ does not depend on the original sender of a message in any way, then it can be called by *any* user even if they were not participating in the forwarding path of that message. This is a strong claim since the number of possible senders of any particular message in *Hecate* is now as large as the number of users in the EEMS.

In *Hecate*, $\text{Forge}_{\text{rec}}$ allows users to forge messages by using their own pre-processing tokens and constructing their own franked messages that they send back to themselves. The receiver does not have any more capabilities than any other user, and in particular the sender, without the secret key of the moderator. In other words,

Source's Payload							Forwarder's Payload	Envelope			
x_1	x_2	nonce	pk_e	r	t_1	σ_1	σ_2	envelope of source	com	σ_3	t_2
32B	32B	12B	32B	32B	8B	64B	64B	104B	32B	64B	8B

Table 2.3: The format of a franked message delivered to the receiver, along with sizes in bytes for the implementation in §2.7. A franked message sent by the source is similar, except the envelope does not yet contain σ_3 or t_2 .

$DENM_b^A$

```

1:  $s_1, s_2 \leftarrow \$\mathcal{A}$ 
2:  $k_{mod} \leftarrow \$KGen(s_1)$ 
3:  $(pk_{mod}, sk_{mod}) \leftarrow \$KGen(s_2)$ 
4:  $(pk_{plat}, sk_{plat}) \leftarrow \$KGen(1^n)$ 
5:  $b' \leftarrow \$\mathcal{A}^{ODENM}(k_{mod}, sk_{mod})$ 
6: return  $b'$ 

```

$DENR_b^A$

```

1:  $k_{mod} \leftarrow \$KGen(1^n)$ 
2:  $(pk_{plat}, sk_{plat}) \leftarrow \$KGen(1^n)$ 
3:  $b' \leftarrow \$\mathcal{A}^{ODENR}$ 
4: return  $b'$ 

```

$Forge_{mod}(id_{src}, id_{rec}, m, sk_{mod})$

```

1:  $token := construct_{token}(sk_{mod}, id_{src})$ 
2:  $m_{frank} := construct_{frank}(m, token)$ 
3: return  $m_{frank}$ 

```

$Forge_{rec}(m, state_{rec})$

```

1: fetch token from staterec
2:  $m_{frank} := construct_{frank}(m, token)$ 
3:  $C_{frank} := forge_{eems}(m_{frank}, state_{rec})$ 
4: return  $C_{frank}$ 

```

$O_b^{DENR}(m, id_{src}, id_{rec})$

```

1: if  $b = 0$  :
2:   fetch staterec
3:    $C_{frank} := Forge_{rec}(m, state_{rec})$ 
4:    $m_{frank}' :=$ 
5:      $receive_{amf}(C_{frank}, id_{rec}, time, sk_{plat})$ 
6: else :
7:    $C_{frank} := send_{amf}(m, id_{src}, id_{rec})$ 
8:    $m_{frank}' :=$ 
9:      $receive_{amf}(C_{frank}, id_{rec}, time, sk_{plat})$ 
10: return  $m_{frank}'$ 

```

$O_b^{DENM}(m, id_{src}, id_{rec})$

```

1: if  $b = 0$  :
2:    $m_{frank} :=$ 
3:      $Forge_{mod}(id_{src}, id_{rec}, m, sk_{mod})$ 
4:    $C_{frank} := send_{eems}(m_{frank}, id_{rec})$ 
5:    $m_{frank}' :=$ 
6:      $receive_{amf}(C_{frank}, id_{rec}, time, sk_{plat})$ 
7: else :
8:    $C_{frank} := send_{amf}(m, id_{src}, id_{rec})$ 
9:    $m_{frank}' :=$ 
10:     $receive_{amf}(C_{frank}, id_{rec}, time, sk_{plat})$ 
11: return  $m_{frank}'$ 

```

Figure 2.6: The security games for Deniability with respect to the Receiver ($DENR_b^A$) and Moderator ($DENM_b^A$).

the only thing that a receiver can do is construct the franked message themselves. In **Hecate**, tokens and franked messages are not bound to the sender’s long term key material and the origin of franked messages as a result is indistinguishable without the secret key of the moderator.

In **Forge_{mod}** on the other hand, the moderator can forge messages by using their secret key to produce tokens for any user identity of their choosing, constructing franked messages on their behalf and sending the resulting message to the receiver. This is again a result of how **Hecate** does not bind the user’s long term key material to a franked message.

Formalizing moderator deniability

Theorem 2.6.1. *Hecate is deniable against a moderator. Any adversary \mathcal{A} has advantage $\text{Adv}_{\text{Hecate}}^{\text{denm}}(\mathcal{A}) = 0$.*

The essence of Thm. 2.6.1 is the claim that **Hecate**’s real send routine is indistinguishable from the moderator’s forgery. Intuitively, **Hecate** achieves moderator deniability because **Hecate** implements algorithms **TGen**, **Frank** and **Forward** without ever using the user’s long term key materials and instead relies on ephemeral keys that the moderator generated. Additionally, the preprocessing token relies on an encryption and signature by the moderator in a way that is not directly bound to the message. This claim holds even against a distinguisher who also has the moderator’s secret key – that is, if the moderator chooses to leak their own keys in an attempt to convince the rest of the world about the actions of a sender.

Proof of Thm 2.6.1. We show via a series of hybrids that $\text{DENM}_0^{\mathcal{A}} \stackrel{\epsilon}{\approx} \text{DENM}_1^{\mathcal{A}}$ as shown in Figure 2-6. This effectively boils down to showing that $\text{O}_0^{\text{DENM}} \stackrel{\epsilon}{\approx} \text{O}_1^{\text{DENM}}$, and hence that send_{amf} is computationally indistinguishable from **Forge_{mod}** (in Fig. 2-6)

$\text{send}_0(n)$

```

1 : statesrc := retrievestate(idsrc)
2 : fetch token from statesrc
3 : mfrank := Frank(statesrc, m, idrec, token)
4 : Cfrank := sendeems(mfrank)
5 : return Cfrank

```

$\text{send}_1(n)$

```

constructtoken(kmod, idsrc)
mfrank := Frank(statesrc, m, idrec, token)
Cfrank := sendeems(mfrank)
return Cfrank

```

$\text{send}_2(n)$

```

constructtoken(kmod, idsrc)
mfrank := constructfrank(m, token)
Cfrank := sendeems(mfrank)
return Cfrank

```

Figure 2-7: The hybrid steps modifying send_{amf} in the Moderator Deniability game

and $\text{send}_{\text{eems}}$. Figure 2-7 shows the sequence of hybrid steps here, starting with the existing send_{amf} subroutine as Game_0 .

Game_1 : In send_{amf} , we replace “*fetch token from state_{src}*” with the moderator token construction method $\text{construct}_{\text{token}}$ on the source’s id id_{src} . We can do so because the moderator in *Hecate* does not require any information from a user id_{src} in order to construct a token on their behalf. The adversary cannot observe the authentication that occurs between the sender and the moderator since the oracle is acting on behalf of the moderator in this game.

Game_2 : In send_{amf} , we can disregard the state passed to *Frank* and replace it with its instantiation $\text{construct}_{\text{frank}}$. Similarly to $\text{construct}_{\text{token}}$ (and hence *TGen*), $\text{construct}_{\text{frank}}$ does not require the state of the sender to construct the franked message.

Notice that the resulting game from the prior series of hybrid has transformed send_{amf} to look exactly like $\text{Forge}_{\text{mod}}$. The resulting game is identical to DENM_0^A ,

where only the branch corresponding to $b = 0$ is executed. \square

Formalizing receiver deniability

Theorem 2.6.2. *Hecate is deniable against a malicious receiver. Concretely, for any PPT adversary \mathcal{A} , there exist PPT adversaries \mathcal{A}' and \mathcal{A}'' such that:*

$$\text{Adv}_{\text{Hecate}}^{\text{denr}}(\mathcal{A}) \leq \text{Adv}_{\mathcal{E}}^{\text{eemsdeniability}}(\mathcal{A}') + \text{Adv}_{\mathcal{E}}^{\text{enccpa}}(\mathcal{A}'').$$

That is: Hecate's deniability reduces to the deniability and CPA security properties of the underlying EEMS scheme \mathcal{E} .

The main difference with moderator deniability is that the adversary has to perform a forgery without the secret keys of the moderator. Intuitively, a forger can use her own tokens to create a franked message of her choosing and claim that it came from another source. Users with no access to the moderator's secret key should not be able to verify her claim without breaking the underlying encryption schemes.

Proof of Thm. 2.6.2. We show via a series of hybrids that $\text{DENR}_0^{\mathcal{A}} \stackrel{c}{\approx} \text{DENR}_1^{\mathcal{A}}$, and more precisely show that $\text{O}_0^{\text{DENR}} \stackrel{c}{\approx} \text{O}_1^{\text{DENR}}$, and hence that send_{amf} is computationally indistinguishable from $\text{Forge}_{\text{rec}}$ (Figure 2.6).

Game₀: We start with O_1^{DENR} , we focus on the if-else branch corresponding to $b = 1$ since its the only one executed. We can disregard the other branch.

Game₁: In O_1^{DENR} , we replace $\text{state}_{\text{src}}$ in the send_{amf} subroutine with $\text{state}_{\text{rec}}$ (lines 1, 2, 4). This is equivalent to replacing send_{amf} with its instantiation $\text{construct}_{\text{frank}}$. Since the adversary does not have access to the moderator's secret key, then \mathcal{A} has

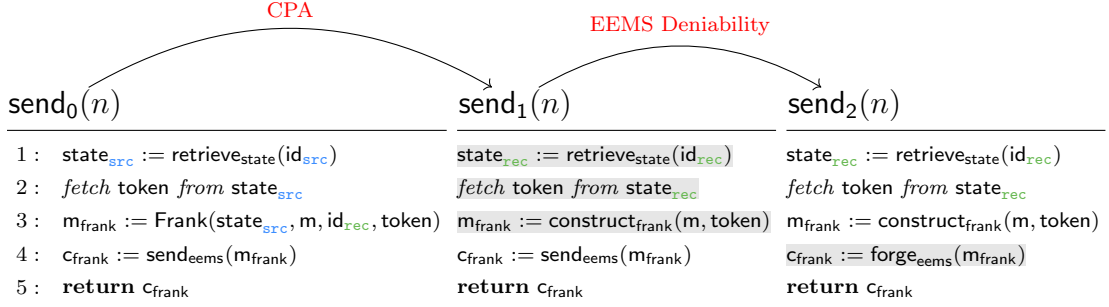


Figure 2-8: The hybrid steps modifying send_{amf} in the Receiver Deniability game

negligible advantage in distinguishing between different x_1 values in the constructed pre-processing token and the franked message. Specifically, they cannot distinguish between an encryption of id_{src} and id_{rec} without breaking the CPA security of the symmetric key encryption scheme used by the moderator during preprocessing. We formally show this by constructing an adversary \mathcal{B} that can break the CPA security game $\text{ENC}_{\text{cpa}}^A$. When adversary \mathcal{A} queries oracle O_1^{DENR} , \mathcal{B} queries $\text{O}_{\text{cpa}}^{\text{enc}}$ with id_{src} and id_{rec} , constructs the franked message with the resulting cipher-text. When \mathcal{A} submits a choice bit b , \mathcal{B} returns the same bit to $\text{ENC}_{\text{cpa}}^A$ and succeeds if and only if \mathcal{A} can distinguish between Games 0 and 1.

Game₂: We replace $\text{send}_{\text{eems}}$ with $\text{forge}_{\text{eems}}$ since the underlying EEMS provides receiver deniability and the receiver hence can forge a channel with themselves.

The resulting game is identical to DENR_0^A , where only the branch corresponding to $b = 0$ is executed. □

2.6.2 Anonymity

Loosely speaking, the anonymity properties that we consider in this work restrict the moderator and any client from learning the metadata about the senders, receivers, and forwarders of messages that are transmitted between other people. We make no claims here about the level of anonymity provided by the underlying network environment (e.g., using sealed sender or Tor). Instead, our goal is to capture that the cryptography does not weaken any anonymity guarantees that happen to be provided by the underlying environment. Put another way: if we assume that the underlying network provides perfect anonymity, we examine the amount of metadata that each entity can learn through the abuse reporting system alone.

Anonymity with respect to the receiver

First, anonymity with respect to the receiver guarantees that *receivers should not be able to learn any other member of the forwarding path of a message beyond their direct neighbors*. For this security property, we assume that senders and forwarders of a message are honest and wish to hide themselves from non-neighboring recipients in the presence of an honest moderator.

We model this property in the ANONR_b^A game. The adversary can send and forward messages between parties using $\mathcal{O}_{\text{anon},b}^{\text{send}}$, $\mathcal{O}_{\text{anon},b}^{\text{fwd}}$ and $\mathcal{O}_{\text{anon}}^{\text{deliver}}$ and is provided with the resulting franked message $\mathbf{m}_{\text{frank}}$.

In this game, we do not attempt to hide chat participants from one another. To that end, both $\mathcal{O}_{\text{anon},b}^{\text{send}}$ and $\mathcal{O}_{\text{anon},b}^{\text{fwd}}$ call $\text{check}_{\text{topology}}$ to ensure that the adversary provided the same pairs of senders and recipients when either of the provided receivers is corrupted. Without this check, \mathcal{A} would trivially win the game by inspecting which of their provided correspondents sent the message or who received it. Additionally, $\text{check}_{\text{topology}}$ ensures that $\mathcal{O}_{\text{anon},b}^{\text{send}}$ and $\mathcal{O}_{\text{anon},b}^{\text{fwd}}$ handle messages relayed from honest

$$\text{ANONR}_b^A$$

```

1 :  $s \leftarrow \mathcal{A}$ 
2 :  $k_{\text{mod}} \leftarrow \mathcal{KGen}(1^n)$ 
3 :  $(pk_{\text{mod}}, sk_{\text{mod}}) \leftarrow \mathcal{KGen}(1^n)$ 
4 :  $(pk_{\text{plat}}, sk_{\text{plat}}) \leftarrow \mathcal{KGen}(s)$ 
5 :  $b' := \mathcal{A}_{b}^{\text{send}, \text{O}_b^{\text{fwd}}, \text{O}^{\text{deliver}}}(sk_{\text{plat}})$ 
6 : return  $b'$ 

```

$$\text{ANONM}_b^A$$

```

1 :  $s_1, s_2, s_3 \leftarrow \mathcal{A}$ 
2 :  $k_{\text{mod}} \leftarrow \mathcal{KGen}(s_1)$ 
3 :  $(pk_{\text{mod}}, sk_{\text{mod}}) \leftarrow \mathcal{KGen}(s_2)$ 
4 :  $(pk_{\text{plat}}, sk_{\text{plat}}) \leftarrow \mathcal{KGen}(s_3)$ 
5 :  $b' := \mathcal{A}_{b}^{\text{fwd}, \text{O}^{\text{deliver}}}(k_{\text{mod}}, sk_{\text{mod}}, sk_{\text{plat}})$ 
6 : return  $b'$ 

```

$$\text{O}^{\text{deliver}}(c_{\text{frank}}, id_{\text{rec}}, \text{time}_{\text{plat}}, sk_{\text{plat}})$$

```

1 :  $m_{\text{frank}} := \text{receive}_{\text{amf}}(c_{\text{frank}}, id_{\text{rec}}, \text{time}_{\text{plat}}, sk_{\text{plat}})$ 
2 :  $R := R \cup \langle m_{\text{frank}}, id_{\text{rec}}, id_{\text{rec}} \rangle$ 
3 : return  $m_{\text{frank}}$ 

```

$$\text{check}_{\text{received}}(m_{\text{frank}}, id_{\text{rec0}}, id_{\text{rec1}})$$

```

1 : if  $\langle m_{\text{frank}}, id_{\text{rec0}}, id_{\text{rec1}} \rangle \notin R$  :
2 :   return  $\perp$ 
3 : return 1

```

$$\text{check}_{\text{topology}}(\langle id_{\text{src0}}, id_{\text{rec0}} \rangle, \langle id_{\text{src1}}, id_{\text{rec1}} \rangle)$$

```

1 : if  $id_{\text{src0}} \vee id_{\text{src1}} \in \text{corrupted}$  :
2 :   return  $\perp$ 
3 : if  $id_{\text{rec0}} \vee id_{\text{rec1}} \in \text{corrupted}$  :
4 :   if  $id_{\text{rec0}} \neq id_{\text{rec1}} \vee id_{\text{src0}} \neq id_{\text{src1}}$  :
5 :     return  $\perp$ 
6 : return true

```

$$\text{O}_{\text{anon}, b}^{\text{fwd}}(m_{\text{frank}}, \langle id_{\text{fwd0}}, id_{\text{rec0}} \rangle, \langle id_{\text{fwd1}}, id_{\text{rec1}} \rangle, \text{time}_{\text{plat}}, sk_{\text{plat}})$$

```

1 : if  $\text{check}_{\text{topology}}(\langle id_{\text{fwd0}}, id_{\text{rec0}} \rangle, \langle id_{\text{fwd1}}, id_{\text{rec1}} \rangle) = \perp$  :
2 :   return  $\perp$ 
3 : if  $\text{check}_{\text{received}}(m_{\text{frank}}, id_{\text{rec0}}, id_{\text{rec1}}) = \perp$  :
4 :   return  $\perp$ 
5 :  $c_{\text{frank}} = \text{fwd}_{\text{amf}}(m_{\text{frank}}, id_{\text{src}}, id_{\text{rec}})$ 
6 :  $m_{\text{frank}}' := \text{receive}_{\text{amf}}(c_{\text{frank}}, id_{\text{rec}}, \text{time}, sk_{\text{plat}})$ 
7 :  $R := R \cup \langle m_{\text{frank}}', id_{\text{rec0}}, id_{\text{rec1}} \rangle$ 
8 : return  $m_{\text{frank}}'$ 

```

$$\text{O}_{\text{anon}, b}^{\text{send}}(m, \langle id_{\text{src0}}, id_{\text{rec0}} \rangle, \langle id_{\text{src1}}, id_{\text{rec1}} \rangle, \text{time}_{\text{plat}}, sk_{\text{plat}})$$

```

1 : if  $\text{check}_{\text{topology}}(\langle id_{\text{src0}}, id_{\text{rec0}} \rangle, \langle id_{\text{src1}}, id_{\text{rec1}} \rangle) = \perp$  :
2 :   return  $\perp$ 
3 :  $c_{\text{frank}} = \text{send}_{\text{amf}}(m, id_{\text{src}}, id_{\text{rec}})$ 
4 :  $m_{\text{frank}} := \text{receive}_{\text{amf}}(c_{\text{frank}}, id_{\text{rec}}, \text{time}, sk_{\text{plat}})$ 
5 :  $R := R \cup \langle m_{\text{frank}}, id_{\text{rec0}}, id_{\text{rec1}} \rangle$ 
6 : return  $m_{\text{frank}}$ 

```

Figure 2-9: The security games for Anonymity with respect to the Receiver and Moderator.

senders and forwarders. $\mathcal{O}_{\text{anon}}^{\text{deliver}}$ on the other hand allows \mathcal{A} to send franked messages from corrupted nodes to an honest receiver. In $\mathcal{O}_{\text{anon},b}^{\text{fwd}}$, we also check that the queried $\mathbf{m}_{\text{frank}}$ was initially received by either of the provided forwarders using the $\text{check}_{\text{received}}$ method in order to model the actual behavior of messages forwarders. Both $\mathcal{O}_{\text{anon},b}^{\text{fwd}}$ and $\mathcal{O}_{\text{anon}}^{\text{deliver}}$ check that the provided franked message are consistent with the recipient's state by calling $\forall f$ in the $\text{receive}_{\text{amf}}$ subroutine, thus eliminating any trivial wins that arise from malformed messages.

All three oracles, along with the corruption oracles $\mathcal{O}^{\text{corrupt}}$ and $\mathcal{O}^{\text{request}}$, allow the adversary to adaptively build any two message paths of their choice (modulo the topological restrictions) and receive the transcript of the chosen path, effectively encompassing the full power of a malicious recipient that may intercept messages along the path. The adversary is tasked with guessing which of the two message paths, with honest sources/roots, was chosen by the game. If they fail to distinguish between them, then they would have failed to determine the original sender of that message and the anonymity of that user and honest forwarder along the path is preserved.

Theorem 2.6.3. *Hecate is anonymous with respect to the receiver. For any PPT adversary \mathcal{A} , there exists an adversary \mathcal{A}' that can win the chosen plaintext attack game with advantage $\text{Adv}_{\text{Hecate}}^{\text{anonr}}(\mathcal{A}) \leq \text{Adv}_{\text{Hecate}}^{\text{enc}_{\text{cpa}}}(\mathcal{A}')$.*

Informally, this theorem holds because the preprocessing tokens in **Hecate** only contain any information about the original sender's identity in encrypted form; without access to the moderator's secret key, a receiver can't distinguish between tokens that originate from different senders. Additionally, **Hecate** stores no information about forwarders of a message at all, thereby guaranteeing their anonymity as well. We provide a rigorous proof of this theorem below.

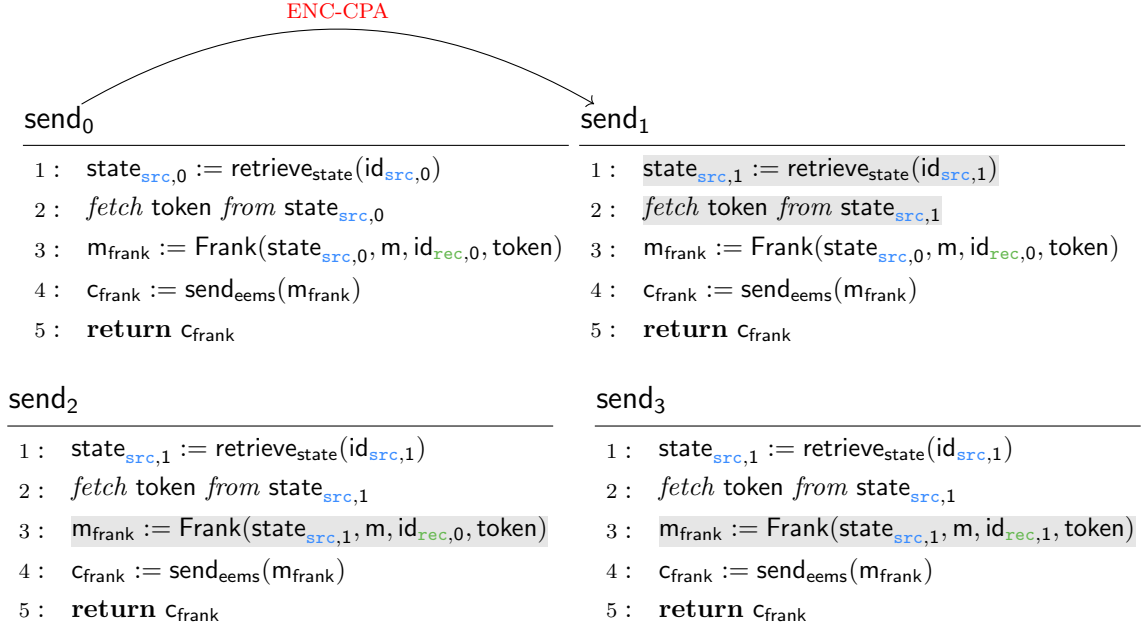


Figure 2-10: The hybrid steps of send_{amf} in the ANONR_b^A game. We refer the reader to ANONM_b^A hybrids for fwd_{amf} 's hybrids (We omit the final hybrid from this figure for ease of presentation).

Proof of Thm. 2.6.3. At a high level, **Hecate** guarantees sender anonymity for the same reason it achieves receiver deniability: with no access to the moderator's secret key, message recipients cannot tell who the originator of a message is without breaking the underlying encryption scheme. Additionally, since the commitment scheme used for envelope commitments is hiding, then access to the platform's secret key does not reveal anything about the sender of a message. On the other hand, forwarding franked messages in **Hecate** does not utilize the forwarder's state or identity. In other words, no attribute in any franked message can be traced back to a forwarder guaranteeing forwarder anonymity.

We show via a series of hybrids that $\text{ANONR}_0^A \stackrel{c}{\approx} \text{ANONR}_1^A$.

Game₀: We start with ANONR_b^A with $b = 0$.

Game₁: We replace $\text{state}_{\text{src},0}$ with $\text{state}_{\text{src},1}$ in the send_{amf} subroutine that is called by $\mathcal{O}_{\text{anon},b}^{\text{send}}$. The moderator in *Hecate* binds a sender to a token by encrypting their identity on line 3 in $\text{construct}_{\text{token}}$ (Figure 2.4) and generating the sub-token x_1 . Without the moderator's key, the adversary cannot decrypt the identity of the sender within x_1 in the token constructed in send_{amf} (Figure 2.5) on line 2, without breaking the CPA security of the underlying encryption scheme. The formalism here is similar to Game 1 of $\text{DEN}_{\text{rec}}^A$.

Game₂: We replace $\text{state}_{\text{src},0}$ with $\text{state}_{\text{src},1}$ in *Frank* in the send_{amf} subroutine that is called by $\mathcal{O}_{\text{anon},b}^{\text{send}}$. The user's state and long term keys are never used during the construction of the franked message via $\text{construct}_{\text{frank}}$ in *Hecate*. m_{frank} is only bound to a particular user by x_1 which we have discussed and handled in the previous hybrid.

Game₃: We replace $\text{id}_{\text{rec}0}$ with $\text{id}_{\text{rec}1}$ in both $\mathcal{O}_{\text{anon},0}^{\text{send}}$ and $\mathcal{O}_{\text{anon},0}^{\text{fwd}}$. In $\mathcal{O}_{\text{anon}}^{\text{send}}$, $\text{check}_{\text{topology}}$ enforces that $\text{id}_{\text{src}0} = \text{id}_{\text{src}1}$ and $\text{id}_{\text{rec}0} = \text{id}_{\text{rec}1}$ when either of the receivers is corrupted (and similarly for $\mathcal{O}_{\text{anon}}^{\text{fwd}}$). If both receivers are honest, then we can make this replacement because *Hecate* does not use any information related to the receiver of a message when constructing or forwarding a franked message (see $\text{construct}_{\text{frank}}$ and $\text{construct}_{\text{fwd}}$ respectively) and the adversary cannot hence distinguish between both Games 2 and 3.

We have shown that $\mathcal{O}_{\text{anon},0}^{\text{send}} \stackrel{c}{\approx} \mathcal{O}_{\text{anon},1}^{\text{send}}$ in *Hecate* since both oracles are now identical. The next series of hybrids are similar to the ones we have already seen.

Game₄: We replace $\text{state}_{\text{fwd},0}$ with $\text{state}_{\text{fwd},1}$ in *Forward* in the fwd_{amf} subroutine

that is called by $O_{\text{anon},b}^{\text{fwd}}$. The reason we can do so is two folds: (1) *Hecate* does not use the forwarder states in constructing the franked message, (2) when the receiver of a forwarded message is corrupted, $\text{check}_{\text{topology}}$ enforces that $\text{id}_{\text{fwd},0} = \text{id}_{\text{fwd},1}$ and $\text{id}_{\text{rec}0} = \text{id}_{\text{rec}1}$. In either case, the source or forwarder of a message have to be honest.

The resulting game is identical to ANONR_1^A . We conclude that *Hecate* is sender and forwarder anonymous and the advantage of the adversary is equal to that of the $\text{ENC}_{\text{cca}}^A$. □

Second, anonymity with respect to the moderator ensures that *the moderator should not be able to learn members of the forwarding path of a reported message beyond the neighbors of colluding receivers and the reported source*. Here, honest forwarders want to be assured that, when their direct contacts are honest, only their neighboring recipients know that they forwarded a specific message. Since the moderator can directly trace the source of a franked message after receiving it, we can additionally assume that for all intents and purpose the sender of a message in this game is also colluding with them.

Anonymity with respect to the moderator

We show this property via the ANONM_1^A game, where the adversary now only has access to $O_{\text{anon},b}^{\text{fwd}}$, $O_{\text{anon},b}^{\text{deliver}}$, O^{corrupt} , O^{request} oracles. Contrary to the prior property, the adversary now chooses the moderator's secret keys and controls the root of the message path. They must now, as a result, use $O_{\text{anon},b}^{\text{deliver}}$ to deliver messages between the compromised sender and honest recipients. If, by the end of the game, the adversary cannot guess the correct message path chosen by the game then they could not have distinguished between the different honest forwarders provided in each message path. The adversary in this game is strictly stronger than the one in ANONR_b^A because

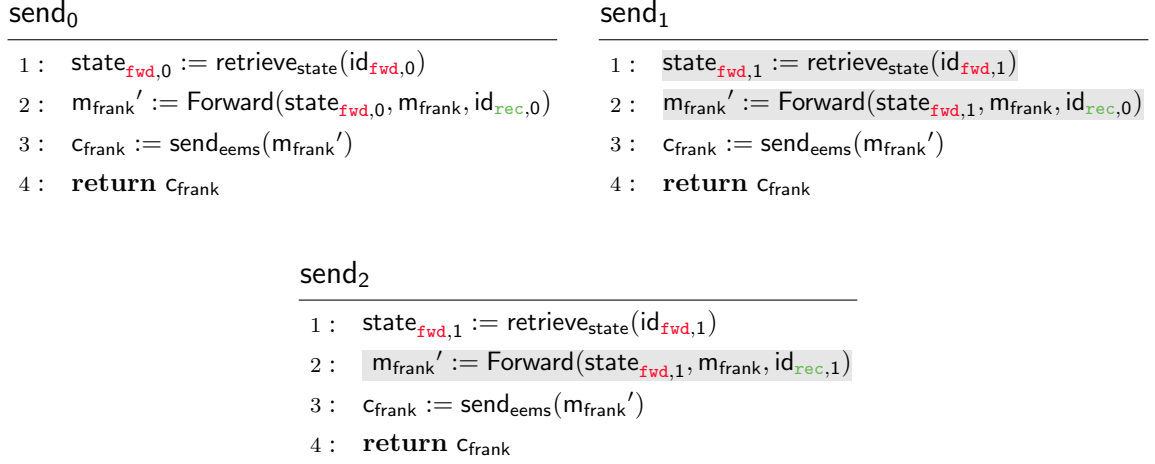


Figure 2.11: The hybrid steps of send_{amf} in the $\text{ANON}_{\text{mod,b}}^A$ game. We refer the reader to the moderator anonymity game for fwd_{amf} 's hybrids.

of the gained source tracing capability from the moderators secret keys, and hence implies the anonymity of forwarders in the presence of malicious receivers and an honest moderator. However, ANONR_b^A independently makes that guarantee because of the way $\text{O}_{\text{anon,b}}^{\text{fwd}}$ handles interactions between honest users. By the end of ANONR_b^A , if the adversary could not guess the message path chosen by the game, then they could not distinguish between honest forwarders with honest neighbors as well.

Theorem 2.6.4. *Hecate is anonymous with respect to the moderator. Any adversary \mathcal{A} has advantage $\text{Adv}_{\text{Hecate}}^{\text{anonm}}(\mathcal{A}) = 0$.*

Proof of Thm. 2.6.4. In this proof we need to show that a corrupted moderator cannot learn the forwarding path of a message beyond senders/forwarders/receivers that she has already corrupted and their neighbors. The reader is referred to the similar receiver anonymity proof for more details.

We can show via a series of hybrids that $\text{ANONM}_0^A \stackrel{c}{\approx} \text{ANONM}_1^A$.

Game₀: We start with ANONM_0^A with $b = 0$.

Game₁: We replace $\text{state}_{\text{fwd},0}$ with $\text{state}_{\text{fwd},1}$ in `Forward` in the fwd_{amf} subroutine that is called by $\mathcal{O}_{\text{fwd},b}^{\text{fwd}}$. The reason we can do so is two folds: (1) `Hecate` does not use the forwarder states in constructing the franked message, (2) when the receiver of a forwarded message is corrupted, $\text{check}_{\text{topology}}$ enforces that $\text{id}_{\text{fwd},0} = \text{id}_{\text{fwd},1}$ and (3) that both forwarders passed to $\mathcal{O}_{\text{anon},0}^{\text{fwd}}$ are honest.

Game₃: We can replace $\text{id}_{\text{rec},0}$ with $\text{id}_{\text{rec},1}$ in the fwd_{amf} and $\text{receive}_{\text{amf}}$ subroutines that are called by $\mathcal{O}_{\text{anon},b}^{\text{fwd}}$. $\text{check}_{\text{topology}}$ enforces that $\text{id}_{\text{rec},0} = \text{id}_{\text{rec},1}$ when either of the receivers is corrupted (and similarly for $\mathcal{O}_{\text{anon},0}^{\text{fwd}}$). If both receivers are honest, then we can make this replacement because `Hecate` does not use any information related to the receiver of a message when constructing or forwarding a franked message (see $\text{construct}_{\text{fwd}}$) and the adversary cannot hence distinguish between both Games 2 and 3.

We can conclude that $\text{ANONM}_0^{\mathcal{A}}$ becomes indistinguishable from $\text{ANONM}_1^{\mathcal{A}}$. \square

Note that our construction and security games do not consider forwarding graphs (i.e. trees with cycles). In those cases, users can identify that the same message was forwarded to them multiple times, a property called *tree linkability* in prior work [203]. Additionally, we allow users (but not the platform or the moderator) to distinguish between a sent and a forwarded message as is the case in several messaging system. We believe that an exciting opportunity for future work is to combine the ideas in this paper with the tree unlinkability scheme by Peale et al. [203].

CONF-FS_b^A <hr style="border: 0.5px solid black;"/> 1 : $s_1, s_2 \leftarrow \mathcal{A}$ 2 : $k_{\text{mod}} \leftarrow \mathcal{KGen}(s_1)$ 3 : $(pk_{\text{mod}}, sk_{\text{mod}}) \leftarrow \mathcal{KGen}(s_2)$ 4 : $F := \emptyset$ 5 : $b' := \mathcal{A}^{O^*}(k_{\text{mod}}, sk_{\text{mod}})$ 6 : return b'	$O_{\text{conf-fs},b}^{\text{fwd}}(m_{\text{frank}0}, m_{\text{frank}1}, id_{\text{fwd}}, id_{\text{rec}})$ <hr style="border: 0.5px solid black;"/> 1 : if $\forall id \in \{id_{\text{fwd}}, id_{\text{rec}}\}, id \in \text{corrupted} :$ 2 : return \perp 3 : $C_{\text{frank}} := \text{fwd}_{\text{amf}}(m_{\text{frank}b}, id_{\text{src}}, id_{\text{rec}})$ 4 : return C_{frank}
$O_{\text{conf-fs},b}^{\text{send}}(m_0, m_1, id_{\text{src}}, id_{\text{rec}})$ <hr style="border: 0.5px solid black;"/> 1 : if $\forall id \in \{id_{\text{src}}, id_{\text{rec}}\}, id \in \text{corrupted} :$ 2 : return \perp 3 : $C_{\text{frank}} := \text{send}_{\text{amf}}(m_b, id_{\text{src}}, id_{\text{rec}})$ 4 : $C := C \cup C_{\text{frank}}$ 5 : return C_{frank}	$O_{\text{conf-fs}}^{\text{decrypt}}(C_{\text{frank}}, id_{\text{rec}})$ <hr style="border: 0.5px solid black;"/> 1 : if $C_{\text{frank}} \in C :$ 2 : return \perp 3 : $m_{\text{frank}} :=$ 4 : $\text{receive}_{\text{amf}}(C_{\text{frank}}, id_{\text{rec}}, \text{time}, sk_{\text{plat}})$ 5 : return m_{frank}

Figure 2-12: The security games for Message Confidentiality. Here, we use O^* to denote $O_{\text{conf-fs},b}^{\text{send}}$, $O_{\text{conf-fs},b}^{\text{fwd}}$, $O_{\text{conf-fs}}^{\text{decrypt}}$, O^{corrupt} and O^{request} .

2.6.3 Message Confidentiality and Forward Secrecy

Message confidentiality dictates that *any party not involved in the creation, reception or reporting of a message should not be able to learn anything about the message*. Moreover, forward secrecy guarantees that *corrupted users should be guaranteed confidentiality of all their messages and interactions prior to the time of compromise*. In this work, we consider the state of users to consist entirely of their key material and their tokens; ergo, Hecate can only guarantee confidentiality for messages that have been securely deleted from the local device prior to the compromise event.

We provide a combined definition of message confidentiality and forward security in Figure 2-12. It guarantees message confidentiality because the CONF-FS_b^A game requires that content moderation does not break CCA security. Additionally, it guarantees forward security because the adversary in the CONF-FS_b^A game is allowed to corrupt any user of their choice, and in particular they can corrupt users who had previously honestly interacted using $O_{\text{conf-fs},b}^{\text{send}}$ and $O_{\text{conf-fs},b}^{\text{fwd}}$. The game requires that the

adversary cannot learn anything about their previously exchanged honest messages, their prior keys, or states.

In this game, we note an important type difference between the franked messages m_{frank} returned by $O_{\text{conf-fs}}^{\text{decrypt}}$ on one hand, and the franked cipher c_{frank} returned by $O_{\text{conf-fs,b}}^{\text{send}}$ and $O_{\text{conf-fs,b}}^{\text{fwd}}$ on the other. $O_{\text{conf-fs,b}}^{\text{send}}$ and $O_{\text{conf-fs,b}}^{\text{fwd}}$ produce a franked cipher that is handed out to the platform for stamping before it gets relayed back to the receiver. In other words, the platform cannot read any part of c_{frank} that is not intended for it. $O_{\text{conf-fs}}^{\text{decrypt}}$ returns the franked message after it has been delivered and hence decrypted by the receiver. If a content moderation scheme does not handle the distinction between the franked message and franked cipher properly, by say appending the id of the sender to the envelope, then the adversary should be able to easily win the game.

Theorem 2.6.5. *Our scheme Hecate is message confidential and forward secure. Concretely, for any PPT adversary \mathcal{A} , there exist PPT adversaries \mathcal{A}' and \mathcal{A}'' such that:*

$$\text{Adv}_{\text{Hecate}}^{\text{conf-fs}}(\mathcal{A}) \leq \text{Adv}_{\text{Hecate}}^{\text{hiding}_{\text{com}}}(\mathcal{A}') + \text{Adv}_{\text{Hecate}}^{\text{enc}_{\text{cca}}}(\mathcal{A}'').$$

Informally, the theorem holds because **Hecate** constructs franked messages by appending tokens to the payload of the message, and by adding a commitment and timestamp to its envelope (see $\text{construct}_{\text{frank}}$ and $\text{stamp}_{\text{time}}$ in Figure 2.4). The tokens are encrypted alongside the plaintext message. The identifying content of the commitment is encrypted, and we rely on the hiding properties of the commitment scheme and the security of the connection that exists between parties and the platform. No part of a **Hecate** franked message can therefore break this security property. We prove this theorem in detail in what follows.

Proof of Thm. 2.6.5. In this proof, we show that **Hecate** does not break the CCA

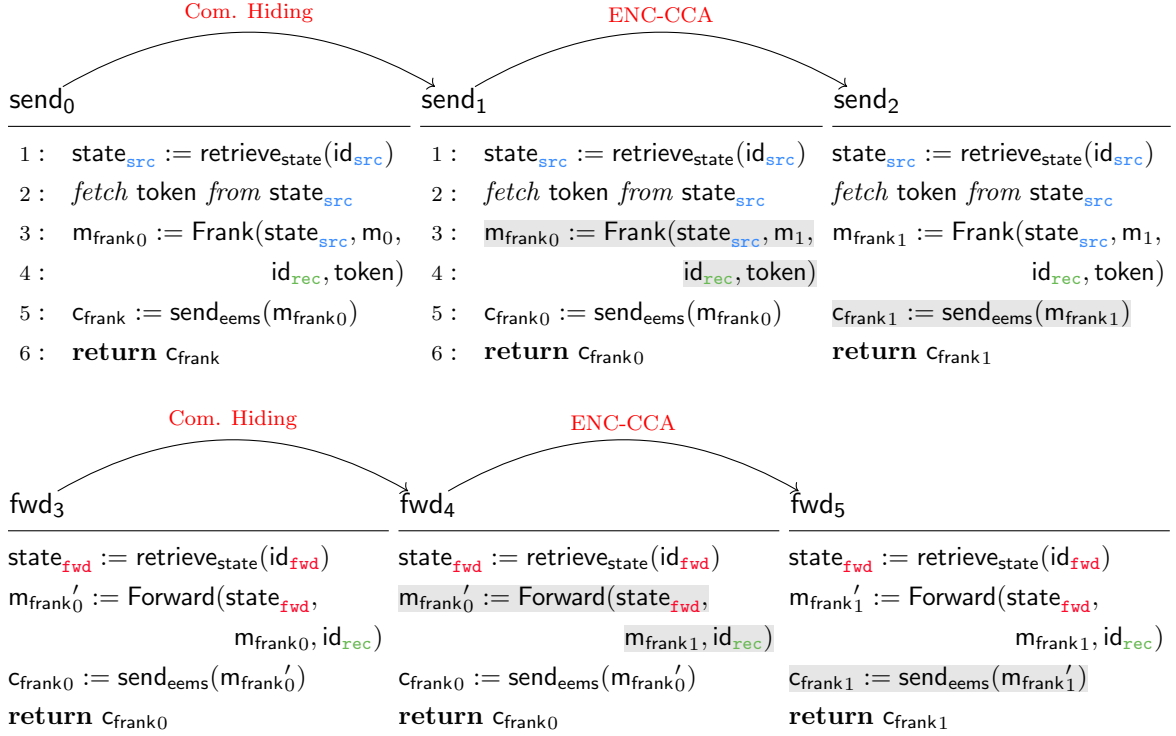


Figure 2-13: The hybrid steps of the CONF-FS_b^A game

security of the underlying cryptographic scheme.

Game₀: We start with CONF-FS₀^A.

Game₁: In send_{amf} (called by O_{conf-fs,b}^{send}), we replace m₀ with m₁ in Frank on line 4. Since the adversary does not have access to r, then they cannot only decommit com with negligible property equal to breaking the secrecy of the commitment scheme. We construct \mathcal{B} that can uses an \mathcal{A} that can distinguish between both games, to break the secrecy of the commitment scheme. When \mathcal{A} calls O_b^{send} on messages m₀ and m₁, \mathcal{B} queries O^{com-hiding} with both messages and uses the returned com to construct the franked cipher on behalf of the moderator and the users c_{frank}. If \mathcal{A} succeeds in picking a choice bit, then \mathcal{B} will pick the same choice bit in Com_{hiding}^A and will win

with at least the same advantage.

Game₂: In send_{amf} (called by $\text{O}_{\text{conf-fs,b}}^{\text{send}}$), we replace $\mathbf{m}_{\text{frank0}}$ with $\mathbf{m}_{\text{frank1}}$ in Frank. Since the adversary does not have access to the source id_{src} 's encryption secret keys, then they can only notice this change with advantage equal to breaking the CCA security $\text{ENC}_{\text{cca}}^{\mathcal{A}}$ of $\text{send}_{\text{eems}}$ and decrypting the contents of $\mathbf{c}_{\text{frank}}$. When \mathcal{A} calls $\text{O}_{\text{conf-fs,b}}^{\text{send}}$ on messages $\mathbf{m}_{\text{frank0}}$ and $\mathbf{m}_{\text{frank1}}$, \mathcal{B} queries $\text{O}_{\text{cca}}^{\text{enc}}$ with both messages and uses the returned cipher text $\mathbf{c}_{\text{frank}}$ to construct the franked message on behalf of the moderator and the users. When \mathcal{A} calls $\text{O}^{\text{decrypt}}$ on ciphertext c , \mathcal{B} behaves similarly and requests the decryption of the cipher-text from $\text{O}_{\text{cca}}^{\text{decrypt}}$. If \mathcal{A} succeeds in picking a choice bit, then \mathcal{B} will pick the same choice bit in $\text{ENC}_{\text{cca}}^{\mathcal{A}}$ and will win with at least the same advantage.

Game₃: In fwd_{amf} (called by $\text{O}_{\text{conf-fs,b}}^{\text{fwd}}$), we replace $\mathbf{m}_{\text{frank0}}$ with $\mathbf{m}_{\text{frank1}}$ in Forward on line 4. When a message is forwarded in Hecate, the commitment com in the original franked message is replaced with a random commitment and the adversary cannot distinguish this change.

Game₄: In fwd_{amf} (called by $\text{O}_{\text{conf-fs,b}}^{\text{fwd}}$), we replace $\mathbf{m}_{\text{frank0}}'$ with $\mathbf{m}_{\text{frank1}}'$ in Forward. Since the adversary does not have access to the source id_{src} 's encryption secret keys, then they can only notice this change with advantage equal to breaking the CCA security of $\text{send}_{\text{eems}}$ and decrypting the contents of $\mathbf{c}_{\text{frank}}$. The proof is similar to the one in Game₁.

The resulting $\text{CONF-FS}_0^{\mathcal{A}}$ is identical to $\text{CONF-FS}_1^{\mathcal{A}}$ and the adversary can only

$\text{ACC}^{\mathcal{A}}$ <hr style="border: 0.5px solid black;"/> 1 : $s \leftarrow \$\mathcal{A}$ 2 : $k_{\text{mod}} \leftarrow \$\text{KGen}(1^n)$ 3 : $(pk_{\text{plat}}, sk_{\text{plat}}) \leftarrow \$\text{KGen}(s)$ 4 : $\text{report} \leftarrow \$\mathcal{A}^{O^*}(sk_{\text{plat}})$ 5 : $(id, \text{time}, m) := \text{Inspect}(m_{\text{frank}}, k_{\text{mod}})$ 6 : if $id \notin \{\perp, \text{corrupted}\} \wedge m \notin M$: 7 : return 1 8 : return 0	$O_{\text{acc}}^{\text{send}}(m, id_{\text{src}}, id_{\text{rec}}, \text{time}_{\text{plat}}, sk_{\text{plat}})$ <hr style="border: 0.5px solid black;"/> 1 : $c_{\text{frank}} := \text{send}_{\text{amf}}(m, id_{\text{src}}, id_{\text{rec}})$ 2 : $m_{\text{frank}} :=$ 3 : $\text{receive}_{\text{amf}}(c_{\text{frank}}, id_{\text{rec}}, \text{time}, sk_{\text{plat}})$ 4 : $F := F \cup m_{\text{frank}}$ 5 : return m_{frank}
$O_{\text{acc}}^{\text{deliver}}(c_{\text{frank}}, id_{\text{rec}}, \text{time}_{\text{plat}}, sk_{\text{plat}})$ <hr style="border: 0.5px solid black;"/> 1 : $m_{\text{frank}} :=$ 2 : $\text{receive}_{\text{amf}}(c_{\text{frank}}, id_{\text{rec}}, \text{time}_{\text{plat}}, sk_{\text{plat}})$ 3 : return m_{frank}	$O_{\text{acc}}^{\text{fwd}}(m_{\text{frank}}, id_{\text{src}}, id_{\text{rec}}, \text{time}_{\text{plat}}, sk_{\text{plat}})$ <hr style="border: 0.5px solid black;"/> 1 : $c_{\text{frank}} := \text{fwd}_{\text{amf}}(m_{\text{frank}}, id_{\text{src}}, id_{\text{rec}})$ 2 : $m_{\text{frank}}' :=$ 3 : $\text{receive}_{\text{amf}}(c_{\text{frank}}, id_{\text{rec}}, \text{time}, sk_{\text{plat}})$ 4 : return m_{frank}'

Figure 2.14: The security games for Accountability. Here, we use O^* to denote $O_{\text{acc}}^{\text{send}}, O_{\text{acc}}^{\text{fwd}}, O_{\text{acc}}^{\text{deliver}}, O_{\text{request}}$ and O_{corrupt} .

win the $\text{CONF-FS}_b^{\mathcal{A}}$ game with advantage equal to twice the advantage of breaking the CCA security of the underlying encryption scheme and the hiding property of the commitment scheme. □

2.6.4 Unforgeability and Accountability

These properties describe a scheme's ability to *bind senders to well-formed messages while guaranteeing that no user can be accused of sending a message that they did not send*. They go hand-in-hand because well-formed messages are necessarily bound to their original sender and cannot be attributed to anyone else. Note that these properties should hold with respect to an honest moderator who handles source tracing reported messages. Ergo, in the unforgeability and accountability game, the adversary attempts to create a message and fool the moderator into believing that it came from a different user.

We model accountability via the $\text{ACC}^{\mathcal{A}}$ security game in Figure 2.14. The adversary starts by making polynomially many queries to O^{send} , O^{fwd} , $\text{O}^{\text{deliver}}$, $\text{O}^{\text{corrupt}}$, and $\text{O}^{\text{request}}$ that collectively allow \mathcal{A} to build any message path of their choice and receive the resulting franked messages at each node within that path. Afterward, the adversary is tasked with producing a franked message that: (1) can be reported back to an existing user (line 5 in $\text{ACC}^{\mathcal{A}}$) (2) traces back to an uncorrupted party (line 6), and (3) was not previously created during the challenge phase (line 6). In producing a message that can pass these predicates, the adversary can effectively produce new messages that can be traced back to other users. Note that who the message traces back to, beyond being an existing honest user, is inconsequential: if the adversary cannot find anyone else to blame them for a message, regardless of who they are, then they cannot avoid accountability.

Theorem 2.6.6. *Hecate holds users accountable. For any PPT adversary \mathcal{A} that makes at most q queries to its O^{send} oracle, there exist PPT adversaries \mathcal{A}' and \mathcal{A}'' such that:*

$$\text{Adv}_{\text{Hecate}}^{\text{acc}}(\mathcal{A}) \leq (q + 1) \cdot \text{Adv}_S^{\text{sig}_{\text{eu-cma}}}(\mathcal{A}') + \text{Adv}_{\mathcal{H}}^{\text{hash}_{\text{coll}}}(\mathcal{A}'').$$

By abuse of notation, in the following proof whenever we apply set inclusion syntax like “ $\text{token} \in \mathbb{T}$ ” to tokens (which contain data and a signature on that data), we only test set membership for the data components of token against the data components of \mathbb{T} , but allow for any valid signature. In other words, we do not require strong unforgeability in our proofs.

Proof of Thm. 2.6.6. We show how the $\text{ACC}^{\mathcal{A}}$ game can only be won in Hecate with negligible probability. We note that the game’s win condition requires a well-formed

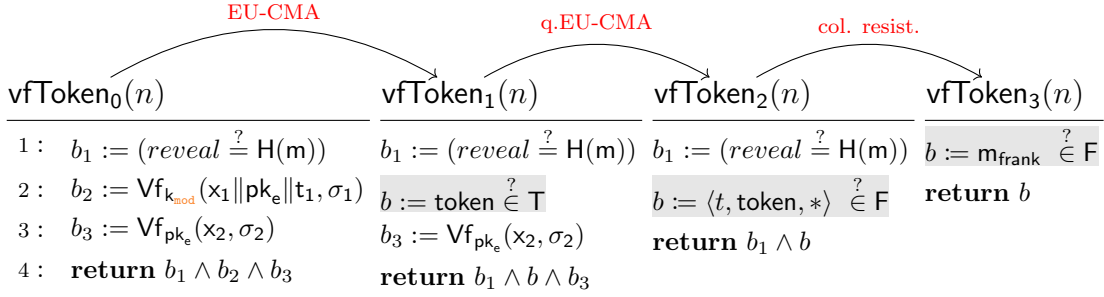


Figure 2.15: The hybrid steps modifying vfToken in the Accountability game

m_{frank} that traces back to an existing id (line 6) and that contains a plain-text message that was not previously submitted and stored in $\text{state}_{\text{chal}}$ (line 6). ACC^A verifies the message integrity via Verify (on line 5), which returns an id in the case of a well formed message and \perp otherwise. Verify is composed of vfMsg (as seen in Figure 2.3 of *Hecate*'s construction) which itself makes three separate calls to vfExp , vfCom and vfToken (line 1-2) based on which it determines the validity of the message. For a franked message m_{frank} to be viable, all three functions must evaluate to true. vfExp and vfCom are envelope commitment verification steps and are hence inconsequential for the accountability property. It's sufficient for the sake of this proof to show that vfToken can never return true in *Hecate*. We handle the other two verification checks in backward security.

In order for vfToken to return true, all three clauses lines (2-4) must also evaluate to true.

Game_0 : This game is equivalent to vfToken , since this is the only method in vfMsg (*Inspect*'s instantiation in *Hecate*) that is relevant for *Hecate*.

Game₁: We replace line 3 of `vfToken` with a new boolean predicate $b := \text{token} \stackrel{?}{\in} \mathbb{T}$ that checks whether a `token` was generated by $\text{O}^{\text{request}}$, and hence by a legitimate call to `TGen`. Note that the only available way for the adversary to retrieve pre-processing tokens is through a call to $\text{O}^{\text{request}}$ with the id of a corrupted party and that every such token is stored in \mathbb{T} on line 5. Since the adversary does not have access to the moderator's secret key, then \mathcal{A} has a negligible advantage in distinguishing this change by breaking the EU-CMA of the underlying signature scheme, forging the signature of the moderator and creating their own token without using $\text{O}^{\text{request}}$. We formally show this by assuming that there exists an adversary \mathcal{A} that can distinguish between Games 0 and 1, and showing how to construct an adversary \mathcal{B} that breaks EU-CMA. We primarily show how \mathcal{B} constructs $\text{O}^{\text{request}}$, all other oracles outputs can be trivially generated by \mathcal{B} in `Hecate` since they are ephemeral and only depend on the pre-processing token and additionally do not depend on the moderator's secret key. When \mathcal{A} requests a pre-processing token, \mathcal{B} locally construct x_1, t_1 and (pk_e, sk_e) , then signs their concatenation by making a query to $\text{Sig}_{\text{eu-cma}}^{\mathcal{A}}$. \mathcal{B} then sends the resulting token to \mathcal{A} and waits until \mathcal{A} submits an m_{frank} . \mathcal{B} additionally checks franked messages delivered using $\text{O}_{\text{acc}}^{\text{deliver}}$ that pass the verification step and the predicates on line 6 in the accountability game. If \mathcal{A} does not fail, \mathcal{B} can strip m_{frank} of everything except the moderator's pre-processing signature σ_1 and submit that to the EU-CMA game. Since this \mathcal{A} is required to submitted a new franked message originating from uncorrupted parties, then the moderator signature in m_{frank} will not correspond to

any of outputs of $\text{Sig}_{\text{eu-cma}}^{\mathcal{A}}$, and \mathcal{B} will win the EU-CMA games when \mathcal{A} does.

Game₂: Let $t := (x_2, \sigma_2)$, i.e. the subset of the franked message constructed during the online stage. We replace and combine lines 3 and 4 with the boolean predicate $b := (\langle t, \text{token}, * \rangle \stackrel{?}{\in} \mathbf{F})$ that checks that the immutable tuple $\langle t, \text{token} \rangle$ is a subset of some franked message in \mathbf{F} . Recall that $\text{O}_{\text{acc}}^{\text{send}}$ is the adversary's only way to instruct honest parties to construct and send messages. All such messages are saved in \mathbf{F} on line 4. Since the adversary does not have access to honest parties' ephemeral keys, then \mathcal{A} has negligible advantage in distinguishing between the original signature verification and the predicate we replaced it with by breaking the EU-CMA of the underlying signature scheme, forging the signature of the an honest party and creating their own franked message originating from that user without using $\text{O}_{\text{acc}}^{\text{send}}$. We formally show this by assuming that there exists an adversary \mathcal{A} that can distinguish between Games 1 and 2, and showing how to construct an adversary \mathcal{B} that breaks EU-CMA. Let q be an upper bound on the number of queries \mathcal{A} can make to $\text{O}_{\text{acc}}^{\text{send}}$. We construct an \mathcal{B} that has access to q different EU-CMA games (with their own $\text{O}_{\text{eu-cma}}^{\text{sign}}$) for q ephemeral key pairs (where $q \in \text{poly}(\lambda)$) and that will try to win at least one of these games. \mathcal{B} starts by sampling and fixing the moderator's secret key and uses it to sign pre-processing tokens. When \mathcal{A} queries $\text{O}_{\text{acc}}^{\text{send}}$, \mathcal{B} picks one of the unused q ephemeral public keys, constructs the pre-processing token for that public key by randomly sampling its secret ephemeral key, and generates the rest of the franked message by asking the $\text{O}_{\text{eu-cma}}^{\text{sign}}$ associated with that public key to sign x_2 . Note

that \mathcal{B} can trivially generate all other fields in the franked message since they have access to the moderator's secret key. Additionally, \mathcal{B} can entirely act on behalf of the moderator when \mathcal{A} calls O^{request} . \mathcal{B} then waits until \mathcal{A} submits an franked messages or delivers a well formed $\mathbf{m}_{\text{frank}}$ using $O_{\text{acc}}^{\text{deliver}}$ that pass predicates on line 6. \mathcal{B} can win the EU-CMA game $\text{Sig}_{\text{eu-cma}}^{\mathcal{A}}$ by stripping that $\mathbf{m}_{\text{frank}}$ of everything except its online signature σ_2 and its ephemeral public key pk_e that it submits to the corresponding $\text{Sig}_{\text{eu-cma}}^{\mathcal{A}}$ game. If \mathcal{A} succeeds at distinguishing Games 1 and 2 using $\mathbf{m}_{\text{frank}}$, then \mathcal{B} will win $\text{Sig}_{\text{eu-cma}}^{\mathcal{A}}$ since $\mathbf{m}_{\text{frank}}$ will need to be a new franked message originating from an uncorrupted party and could not have have been generated by $O_{\text{eu-cma}}^{\text{sign}}$. Since each $\text{Sig}_{\text{eu-cma}}^{\mathcal{A}}$ game is defined by a separate pair of ephemeral keys generated i.i.d., then these games are independent of one another and by the union bound the advantage of the adversary is at most equal to $q \times \text{Adv}_{\mathcal{A}}^{\text{sig}_{\text{eu-cma}}}(n)$.

Game₃: Finally, we can replace and combine all three lines in `vfToken` with the boolean predicate $b := \langle \text{token}, t, m \rangle \stackrel{?}{\in} \mathbf{M}$, which is in essence equivalent to $b := \mathbf{m}_{\text{frank}} \stackrel{?}{\in} \mathbf{M}$. The adversary can distinguish the change made with negligible advantage equal to the likelihood of finding a collision $m' \notin \mathbf{M}$ of the collision resistant hash function H , such that $\exists \mathbf{m}_{\text{frank}}.m \in \mathbf{M}, \text{H}(m) = \text{H}(m')$. Let's assume that there exists an adversary \mathcal{A} that can distinguish between Games 2 and 3. We construct an adversary \mathcal{B} that will try to win the collision resistance hash game. When \mathcal{A} queries $O_{\text{acc}}^{\text{send}}$, \mathcal{B} requests the hash of \mathbf{m} from O^{hash} of the collision resistance hash game, then constructs the rest of the franked message honestly. Note that all other oracles can be run by \mathcal{B}

since they act on behalf of the moderator and honest users. \mathcal{B} then waits until \mathcal{A} submits an franked messages or delivers a well formed $\mathbf{m}_{\text{frank}}$ using $\mathbf{O}_{\text{acc}}^{\text{deliver}}$ that pass predicates on line 6. If \mathcal{A} succeeds at distinguishing Games 2 and 3 using $\mathbf{m}_{\text{frank}}$, then \mathcal{B} will win the collision resistance hash game since the only way \mathcal{A} can distinguish between both games is if finds a collision \mathbf{m}' of $\mathbf{H}(\mathbf{m})$ that was not stored in \mathbf{M} .

Now notice that there can be no franked message $\mathbf{m}_{\text{frank}}$ that can satisfy the hybrid predicate $\mathbf{m}_{\text{frank}} \stackrel{?}{\in} \mathbf{M}$ and the winning condition of the game $\text{id} \notin \{\perp, \text{corrupted}\} \wedge \mathbf{m}_{\text{frank}} \cdot \mathbf{m} \notin \mathbf{M}$, since we have shown that the adversary has to necessarily submit a stored franked message in \mathbf{M} . The adversary cannot therefore construct a franked message $\mathbf{m}_{\text{frank}}$ that can win this game. We can reduce the attacker's advantage in winning the accountability game to the sum of its advantage in breaking the collision-resistance or $(q + 1)$ the EU-CMA games. \square

2.6.5 Backward Security

Backward Security requires that *an adversary who controlled the state and keys of a device pre-compromise should not be able to benefit from them after device recovery*. In particular, the adversary should be unable to craft new messages from a recovered user or claim that during-compromise messages were sent out (not forwarded) after the compromise period.

This kind of property is not inherently handled in the context of abuse reporting by the traditional notion of backward secrecy of end to end encrypted messengers because of the existence of tokens that can live indefinitely in the system. Abuse

$\text{BAC}_{\delta}^{\mathcal{A}}$ <hr/> 1 : $k_{\text{mod}} \leftarrow \$ \text{KGen}(1^n)$ 2 : $(pk_{\text{plat}}, sk_{\text{plat}}) \leftarrow \$ \text{KGen}(1^n)$ 3 : $done \leftarrow \mathcal{A}^{O^*}$ 4 : $T_{\text{pre}} := T$ 5 : $corrupted := \emptyset, F := \emptyset$ 6 : $recovery_{\text{start},t} := \text{global}_t$ 7 : $m_{\text{chal}} \leftarrow \$ \mathcal{M}$ 8 : $\text{global}_t := \text{global}_t + \delta$ 9 : $recovery_{\text{delay},t} := \text{global}_t$ 10 : $m_{\text{frank}} \leftarrow \mathcal{A}^{O^*}(m_{\text{chal}})$ 11 : if $\text{check}_{\text{report}}(m_{\text{frank}})$: 12 : return 1 13 : return 0	$O_{\text{bs}}^{\text{deliver}}(c_{\text{frank}}, id_{\text{rec}})$ <hr/> 1 : $m_{\text{frank}} :=$ 2 : $\text{receive}_{\text{amf}}(c_{\text{frank}}, id_{\text{rec}}, \text{time}_{\text{plat}}, sk_{\text{plat}})$ 3 : return m_{frank}
$\text{check}_{\text{report}}(m_{\text{frank}}, m_{\text{chal}})$ <hr/> 1 : $(m, \text{report}) := \text{Verify}(m_{\text{frank}})$ 2 : $(id_{\text{src}}, \text{time}, m) := \text{Inspect}(\text{report})$ 3 : if $id_{\text{src}} \stackrel{?}{\neq} \perp \wedge$ 4 : $(id_{\text{src}}, *) \stackrel{?}{\notin} corrupted \wedge m_{\text{frank}} \notin F :$ 5 : if $m \stackrel{?}{=} m_{\text{chal}} \vee \text{time} > recovery_{\text{delay},t} :$ 6 : return 1 7 : return 0	$O_{\text{bs}}^{\text{send}}(m, id_{\text{src}}, id_{\text{rec}})$ <hr/> 1 : $c_{\text{frank}} := \text{send}_{\text{amf}}(m, id_{\text{src}}, id_{\text{rec}})$ 2 : $m_{\text{frank}} := \text{receive}_{\text{amf}}(c_{\text{frank}}, id_{\text{rec}}, \text{time}, sk_{\text{plat}})$ 3 : $F := F \cup m_{\text{frank}}$ 4 : return m_{frank}
	$O_{\text{bs}}^{\text{fwd}}(m_{\text{frank}}, id_{\text{fwd}}, id_{\text{rec}})$ <hr/> 1 : $c_{\text{frank}} := \text{fwd}_{\text{amf}}(m_{\text{frank}}, id_{\text{fwd}}, id_{\text{rec}})$ 2 : $m_{\text{frank}}' := \text{receive}_{\text{amf}}(c_{\text{frank}}, id_{\text{rec}}, \text{time}, sk_{\text{plat}})$ 3 : return m_{frank}'
	$O_{\text{bs}}^{\text{stamp}}(c_{\text{frank}})$ <hr/> 1 : $c_{\text{stamp}} := \text{Stamp}(c_{\text{frank}})$ 2 : return c_{stamp}

Figure 2.16: The security games for Backward Security, where O^* denotes O^{send} , O^{fwd} , O^{deliver} , O^{stamp} , O^{corrupt} and O^{request} .

reporting in essence requires careful handling of user's recovery post-compromise so that no compromised state can be used to indefinitely hold users accountable for messages that the adversary might have sent.

To codify backward security, we design the game $\text{BAC}_\delta^{\mathcal{A}}$ in Figure 2-16. We provide the adversary with oracle access to $\text{O}^{\text{corrupt}}$, $\text{O}^{\text{request}}$, O^{send} , O^{fwd} , $\text{O}^{\text{deliver}}$ that allow them to perform any possible interaction between users.

Similarly to prior games, the adversary can corrupt users using $\text{O}^{\text{corrupt}}$ and request their pre-processing tokens via $\text{O}^{\text{request}}$. However contrary to prior games, handling time in $\text{BAC}_\delta^{\mathcal{A}}$ is necessary since backward security is a property of the corruption recovery period. In $\text{BAC}_\delta^{\mathcal{A}}$, corruption is not indefinite and we define it with respect to the global time variable global_t . When \mathcal{A} calls $\text{O}^{\text{corrupt}}$, the oracle stores the id of the corrupted user along with the time of corruption at global_t in the set **corrupted**. When \mathcal{A} calls $\text{O}^{\text{request}}$ to request pre-processing tokens for corrupted users, the oracle first checks that the requested user is in **corrupted** at the current global time and increments the global time if that check succeeds. In other words, the global time is incremented each time a pre-processing token is returned to \mathcal{A} , effectively requiring them to re-corrupt users at the new global time. By defining time in this manner, we capture how an adversary can no longer impersonate recovered users and request tokens on their behalf. The adversary can also stamp any franked cipher that they create using O^{stamp} . This effectively allows the adversary to transmit its own messages.

The adversary can additionally send and forward messages between any two users regardless of their corruption status via O^{send} , O^{fwd} and $\text{O}^{\text{deliver}}$. O^{send} in particular grants the adversary the power to request that an honest user creates a new franked message that the oracle stores in the set **F**. Otherwise, with access to $\text{O}^{\text{request}}$, the adversary can create franked messages on their own before sending/forwarding them along. O^{fwd} allows them to forward a franked message between honest users, and

O^{deliver} allows them to send/forward messages from a corrupted user to an honest one. With access to all these oracles, we model an all powerful adversary that can fully control and build a forwarding tree of their choice.

The BAC_δ^A game proceeds in three phases.

The first phase marks the pre-recovery or compromise phase where the adversary is given access to all aforementioned oracles and can interact with users as they see fit (line 3).

When the adversary is done, the second phase begins and the game master sets up the post-recovery period by: (1) sampling a challenge m_{chal} uniformly at random from the message space excluding plain-text messages sent in the first phase (line 7), (2) saving the time at which the first phase ended in $\text{recovery}_{\text{start},t}$ (line 6), (3) and advancing the current time by a grace period delta to mark the delayed beginning of the recovery period $\text{recovery}_{\text{delay},t}$ (line 9).

The final phase marks the post-recovery period when the adversary is given m_{chal} and asked to provide a franked message m_{frank} that is well formed, traces back to an honest user and either: (1) contains the plain-text challenge or (2) is timestamped after the the delayed recovery period, after they are done interacting with the provided oracles. These possibilities provided to the adversary to win the game encompass this work's notion of backward security: the adversary should neither (1) be able to produce a message that they did not think of during compromise, nor should (2) they circulate a message that was not timestamped during the time of compromise.

The first point codifies how backward secrecy is not concerned with messages created prior to the time of recovery and the adversary should not be able to trivially win the game for any such messages. Instead, the challenger forces the adversary to submit a franked message that they could not have thought of pre-recovery by providing them with a plaintext message that they had not seen before.

The later point essentially captures how abuse reporting systems may deem messages that were sent during known data breach periods as unaccountable and possibly invalid. This is especially important in an era where the number of cyber-attack campaigns is on a rise, and where the adversary may attempt to create many messages during the compromise period and delay reporting them or further circulating them until an opportune time after the recovery period begins. This is especially critical in the case where a public official's device is hacked and when the time at which their leaked messages were sent can influence public opinion.

Note that the game master resets both the **corrupted** and **F** sets on line 5 since they are not useful in the first phase when the recovery period has not yet begun. Both sets will allow the game master to evaluate the response of \mathcal{A} with respect to the winning condition. Since backward security is a property of corruption recovery, the winning condition is itself a function of that period. In this essence, we define recovery after some fixed δ has passed on line 9, to model how recovery in practice is not instantaneous and may require a grace period. And finally the game master saves any corrupted token that the adversary has created during the pre-recovery stage (line 4). This allows the challenger to determine the time difference between the token and the time of stamping later.

Theorem 2.6.7. *Hecate is backward secure when the underlying EEMS and communication channel with the moderator are backward secure. For any PPT adversary \mathcal{A} , there exist PPT \mathcal{A}' , \mathcal{A}'' and \mathcal{A}''' such that:*

$$\text{Adv}_{\text{Hecate}}^{\text{bac}}(\mathcal{A}) \leq \text{Adv}_{\text{Hecate}}^{\text{acc}}(\mathcal{A}') + 2 \cdot \text{Adv}^{\text{sig}_{\text{eu-cma}}}(\mathcal{A}'') + \text{Adv}^{\text{binding}_{\text{com}}}(\mathcal{A}''') + \frac{q}{|\mathcal{M}|},$$

where q is the number of queries made to $\text{O}_{\text{bs}}^{\text{deliver}}$, $\text{O}_{\text{bs}}^{\text{send}}$.

By abuse of notation, in the following proof whenever we apply set inclusion syntax

like “ $\text{token} \in \mathbb{T}$ ” to tokens (which contain data and a signature on that data), we only test set membership for the data components of token against the data components of \mathbb{T} , but allow for any valid signature. In other words, we do not require strong unforgeability in our proofs.

Proof of Thm. 2.6.7. The winning condition of the game requires the chosen $\mathbf{m}_{\text{frank}}$ to trace back to an honest user. $\mathbf{m}_{\text{frank}}$ may then either contain the challenge \mathbf{m}_{chal} chosen by the game master, or be timestamped after the recovery period delay $\text{recovery}_{\text{delay},t}$ on line 9. The game requires $\mathbf{m}_{\text{frank}}$ to be traceable by the moderator via `Inspect` in `check_report`. In *Hecate*, `Inspect` is a conjunction of three separate verification steps: `vfExp` that checks the expiry of the different time components of the franked message, `vfCom` that checks the envelope commitment, and `vfToken` which checks all other parts of the franked message and which we discuss in depth in the accountability game.

In this proof we distinguish between the payload of the franked message that we denote by $\mathbf{m}_{\text{frank}}.\text{payload}$ and its envelope $\mathbf{m}_{\text{frank}}.\text{envelope}$. In what follows, $\mathbf{m}_{\text{frank}}$ is equivalent to $(\mathbf{m}_{\text{frank}}.\text{payload}, \mathbf{m}_{\text{frank}}.\text{envelope})$, $\mathbf{m}_{\text{frank}}.\text{envelope}$ is equivalent to $(\text{com}, \sigma_2, t_2)$. We use the regular expression operator $*$ to denote that a field can take any value and is not specified by a specific game reduction. Each game hybrid will allow us to specify different pieces of the eventual franked message.

Game₁: We replace `vfToken` with checking that $\mathbf{m}_{\text{frank}}$'s preprocessing token is either $(\text{token}, *) \stackrel{?}{\in} \mathbb{F}$ or $\text{token} \stackrel{?}{\in} \mathbb{T}$ (where token is a shorthand for $\mathbf{m}_{\text{frank}}.\text{payload}.\text{token}$). The adversary can distinguish this change with negligible probability equal to the: (1) probability of winning the EU-CMA game since the adversary does not have access to

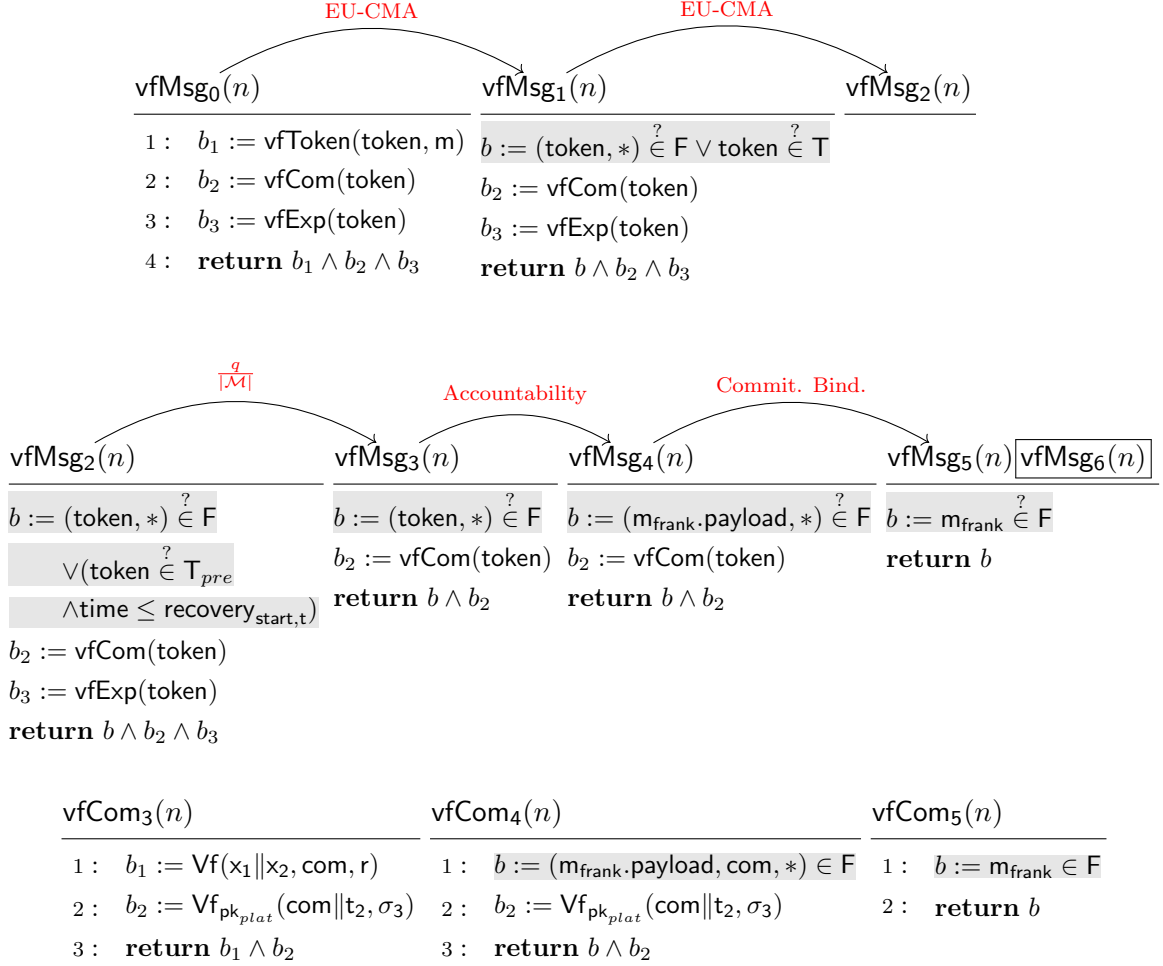


Figure 2-17: The hybrid steps modifying vfMsg and vfCom in the Backward Security game. We use the shorthand token to refer to $m_{\text{frank}}.\text{payload}.\text{token}$ and use the regular expression wildcard operator “*” throughout.

the moderator's secret key and has therefore a negligible chance in constructing well formed tokens for a user of their choice locally. Recall that \mathbb{T} is the set of corrupted users tokens constructed by $\mathcal{O}^{\text{request}}$ and that $(\text{token}, *) \in \mathbb{F}$ refers to the set of honest user tokens constructed by $\mathcal{O}_{\text{bs}}^{\text{send}}$. We refer the reader to Game 1 in the accountability game for more details.

Game₂: We parameterize $\text{expiry} := \delta$ and replace vfExp with $(\text{token}, *) \stackrel{?}{\in} \mathbb{F}$ or $\text{token} \stackrel{?}{\in} \mathbb{T}_{\text{pre}} \wedge \text{time} \leq \text{recovery}_{\text{start}, \text{t}}$ (i.e. that the token was generated prior to the time of recovery and that m_{frank} was time stamped prior to the time of recovery). The adversary can distinguish this change with negligible probability equal to the probability of winning the EU-CMA game since the adversary does not have access to the platform's secret key. vfExp requires that $|\text{t}_2 - \text{t}_1| < \text{expiry}$, i.e. that m_{frank} 's moderator and platform time stamps are within a set expiry time from one another (line 1). The winning condition of the game requires $\text{id}_{\text{src}} \notin \text{corrupted}$. The only possible way for token to have been created by a call $\mathcal{O}^{\text{corrupt}}$ and $\mathcal{O}^{\text{request}}$ must happen prior to the beginning of the recovery time, i.e. $\text{token.t}_1 < \text{recovery}_{\text{start}, \text{t}}$. Recall that users can only be corrupt for one epoch at the current global_t before being considered as honest and requiring the adversary to call $\mathcal{O}^{\text{corrupt}}$ and $\mathcal{O}^{\text{request}}$ if they wish to corrupt them again. This definition of corruption is enforced by $\mathcal{O}^{\text{request}}$: if the id passed is not corrupted at the current global_t , the oracle returns \perp , otherwise it returns the appropriate token and increments global_t . Additionally, since the game master increments the global time by δ after $\text{recovery}_{\text{start}, \text{t}}$ on line 9 in BAC_δ^A then

unless the adversary can forge the platform's signature and create its own timestamps (we refer the reader to the reduction in Game 1 of the accountability game), then if $t_1 > \text{recovery}_{\text{start},t}$, then $\text{envelope.t}_2 > \text{recovery}_{\text{start},t} + \delta$ (where envelope is a shorthand for $m_{\text{frank}}.\text{envelope}$).

Game₃: We replace vfExp with $(\text{token}, *) \stackrel{?}{\in} F$, i.e. we drop the $\text{token} \stackrel{?}{\in} T_{\text{pre}} \wedge \text{time} \leq \text{recovery}_{\text{start},t}$ from the disjunction in Game₂. The adversary can distinguish this game with negligible probability equal to the probability of guessing m_{chal} before the challenger picks it. If $\text{time} \leq \text{recovery}_{\text{start},t}$ then according to the winning condition of the game on line 5, \mathcal{A} must have included m_{chal} in their report. Since the m_{chal} is picked after $\text{recovery}_{\text{start},t}$, then \mathcal{A} must have guessed m_{chal} prior to the time of recovery. They can do so with probability $\frac{q}{|\mathcal{M}|}$, since m_{chal} is sampled uniformly at random from the message sample space.

Game₄: We replace vfToken with checking that $(m_{\text{frank}}.\text{payload}, *) \stackrel{?}{\in} F$, where payload refers to all elements of the franked message that are not on the envelope of the message. The adversary can distinguish this change with negligible probability equal to the advantage of the accountability game since they do not have access to the secret ephemeral keys of users for token constructed with O^{send} . We refer the reader to that proof for more details and not that both Games 1 and 3 are jointly upper bounded by the advantage of the accountability game.

Game₅: In vfCom , we replace line 2 with $b_1 := (m_{\text{frank}}.\text{payload}, \text{com}, *) \in F$. We show that the adversary can distinguish between Games 1 and 2 with negligible ad-

vantage equal to the probability of breaking the binding property of the commitment scheme. Let's assume that there exists an adversary \mathcal{A} that can distinguish both games, we construct an adversary \mathcal{B} that can break the commitment game. Whenever \mathcal{A} calls either O_{bs}^{send} , O_{bs}^{fwd} or O_{bs}^{deliver} , \mathcal{B} constructs/modifies the franked message honestly on behalf of the moderator, the platform and the users and returns the result back to \mathcal{A} . Note that \mathcal{B} stores every single such message. When \mathcal{A} returns a challenge m_{frank} back to \mathcal{B} or successfully delivers an m_{frank} to an honest user that was not previously logged in F using O_{bs}^{deliver} , \mathcal{B} strips the resulting m_{frank} of everything except the commitment and its corresponding decommitment, and submits these values along with the original decommitment stored in F to the commitment game. If \mathcal{A} succeeds then they will have necessarily submitted a decommitment that is different than the original one stored in F and \mathcal{B} will win its corresponding game.

Game₆: In vfCom , we replace line 2 with $b_2 := (m_{\text{frank}}.\text{payload}, \text{com}, t_2, \sigma_2) \in F$, which effectively implies replacing vfCom with $(m_{\text{frank}}.\text{payload}, m_{\text{frank}}.\text{envelope}) \in F$ (i.e. $m_{\text{frank}} \in F$). We can show similarly to Game 1 in the accountability game that, without the platform's secret key, the adversary can distinguish between Games 2 and 3 with negligible advantage equal to the probability of breaking EU-CMA security property of the underlying signature scheme.

Game₇: We replace vfMsg with \perp since we have shown that $m_{\text{frank}} \in F$ and the game's winning condition requires otherwise.

We can therefore conclude that the adversary has negligible advantage in winning

the BAC_δ^A game in *Hecate* equal to:

$$\begin{aligned} \text{Adv}_{\text{Hecate}}^{\text{bac}}(\mathcal{A}) &\leq \text{Adv}_{\text{Hecate}}^{\text{accountability}}(\mathcal{A}) \\ &\quad + 2 \cdot \text{Adv}^{\text{sig}_{\text{eu-cma}}}(\mathcal{A}) \\ &\quad + \text{Adv}^{\text{binding}_{\text{com}}}(\mathcal{A}) \\ &\quad + \frac{q}{|\mathcal{M}|}. \quad \square \end{aligned}$$

where q is the number of queries made to $\text{O}_{\text{bs}}^{\text{deliver}}$, $\text{O}_{\text{bs}}^{\text{send}}$.

Informally, this theorem holds because *Hecate* requires the moderator and platform timestamps to be close to one another. *Hecate* also binds each of these timestamps to the franked message and the source’s identity, but in a confidential way, to ensure that the adversary cannot evade backward secrecy by using unexpired timestamps from other messages.

2.7 Implementation and Evaluation

We implemented *Hecate* as a Rust library that can be used as a back-end by other systems. Our implementation uses Signal’s official platform agnostic API library `libsignal-client` [224] for our encryption, commitment and hashing building blocks. To that effect, we use `libsignal-client`’s implementation of AES-256 GCM for symmetric encryption and HMAC with SHA-256 for commitments. We use the `ed25519-dalek` [94] crate for ed25519 signatures and their associated functions and `SHA256` from the `sha2` crate for our hash functions. Our implementation is open source and available at [140].

In this section, we show experimental results when executing each component of *Hecate* in isolation. Then, we measure the overhead of *Hecate* when integrated into a Signal client.

2.7.1 Performance Cost and Comparison

In this section, we measure the runtimes and transmission sizes for each component within `Hecate` using `Criterion`, a Rust benchmarking suite. We also evaluate prior open-source message franking systems on the same machine and compare them to `Hecate`.

Experimental setup. We ran all experiments on Amazon Web Services in the US East-Ohio Region, using a `t3.small` EC2 virtual machine running Ubuntu 20.04 LTS with 2GB of RAM on a 3.1 GHz Intel Xeon Platinum Processor. Each data point shown is the average of 300 trials, with outliers removed. We chose this machine and number of experiment trials to align with prior work [202, 238].

Hecate communication costs. We list the size of `Hecate`'s franked ciphers in Table 2.3. The sizes in Table 2.3 stem from the fact that our libraries yield 32 byte commitments with 32 byte long commitment randomness, 32 byte ciphertexts, 64 byte signatures, 32 byte symmetric and public keys, 12 byte nonces for symmetric encryption, and 8 byte Unix timestamps. We remark that there exist more compact instantiations of these primitives; our choices were motivated by ease of implementation on top of `libsignal-client`.

Hecate's online runtime. Fig. 2-18 shows the performance of each component of `Hecate` for message sizes ranging from 10 bytes to 10 kB. Overall, the runtime costs remain low especially when compared to the cryptography already required within an end-to-end messaging system (cf. §2.7.2).

Most components require executing a SHA-256 hash function (to calculate x_2) whose runtime is linear in the message size, along with 0-3 digital signature operations whose cost is independent of message size (cf. Table 2.2). As a result, the signature(s)

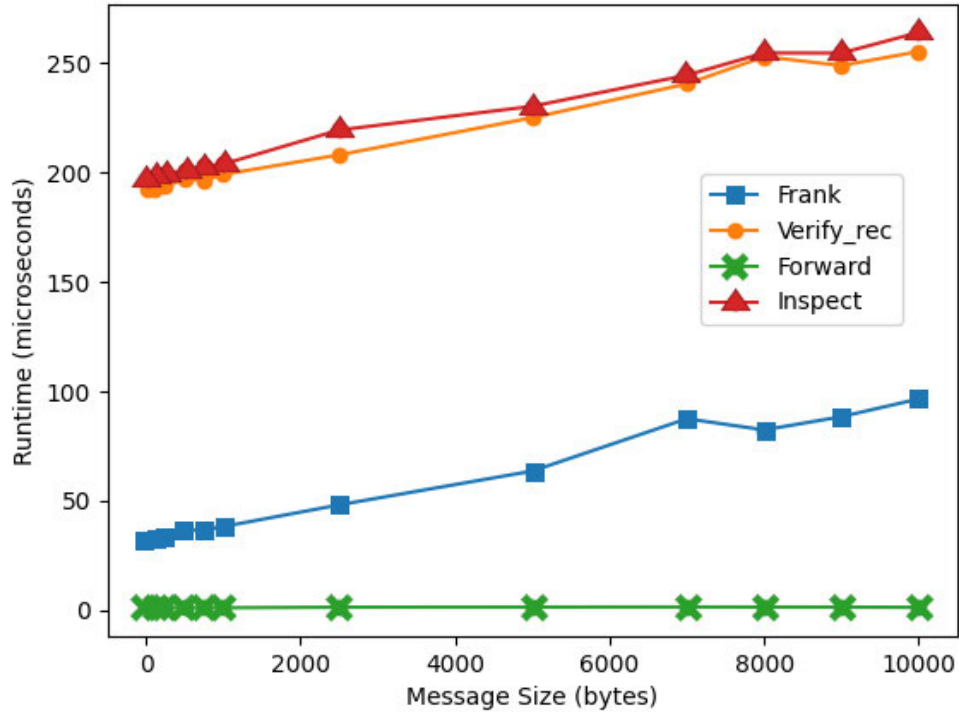


Figure 2-18: Online runtime of Hecate’s components as a function of message size in bytes

dominate the cost for small message sizes and the hash function dominates the cost for large messages. The two costs are balanced at a message size of 7.5 kB, where hashing and digital signing each take about $33\mu s$. Verify and Inspect are slower because they require about $192\mu s$ to verify 3 signatures. On the other hand, Forward, Stamp and TGen all have fast runtimes that are independent of message size. We remark that a forwarder is assumed already to have verified a message at reception time, so its only work during Forward is to move the envelope contents into the payload.

Hecate’s preprocessing cost. We also measure TGen over various batch sizes from 1 to 10,000 tokens. Fig. 2-19 shows the runtime and computational cost based on a rate of 2.09¢ per hour for a t3.small AWS instance at the time of this writing. Our

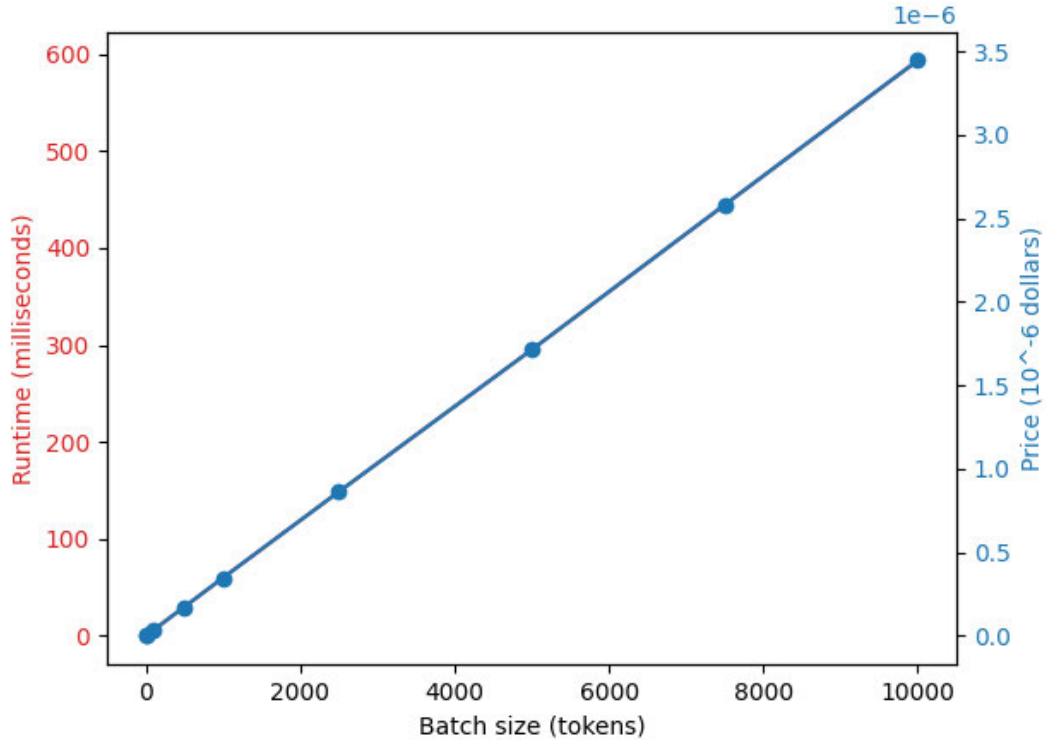


Figure 2-19: Runtime and dollar pricing of pre-processing token generation (TGen) as function of the token batch size.

measurements show that the price of generating a batch of 10^4 tokens is 3.45×10^{-6} USD. Extrapolating to the scale of 10^{11} tokens (the approximate number of messages sent through WhatsApp daily [183]), we estimate the cost of token generation to be 35 cents per day. We also highlight that the moderator does not need to remember these ephemeral signing keys in between preprocessing and reporting; in fact the moderator doesn't require any storage cost at all.

Comparison with prior work. In this section, we compare our Rust implementation with the open-source software by TGLMR [238] and Peale et al. [202]. To ensure a level comparison: we re-ran the benchmarks from prior work [202, 238] on our `t3.small` AWS instance, we only considered the tree-linkable version of Peale et

	Sent	Received	Report
Tyagi et al. [238]	489 B	489 B	489 B
Peale et al. [202]	256 B	320 B	160 B
Hecate	380 B	484 B	380 B

Table 2.4: Communication overhead of **Hecate** and other message franking schemes, in bytes.

	TGen	Frank	Verify	Inspect	Forward	Stamp
Tyagi et al. [238]	–	6339 μs	5461 μs	5939 μs	–	–
Peale et al. [202]	–	8.98 μs	69.56-138.19 μs	73.64 μs	8.46 μs	24.53 μs
Hecate	58.4 μs	38.24 μs	199.15 μs	203.87 μs	1.16 μs	29.17 μs

Table 2.5: Runtimes of message franking schemes, in microseconds, for a message size of 1 kB. The runtime of **Verify** within Peale et al. [202] differs based on whether the message is authored (left) or forwarded (right).

al., and we removed the double ratchet encryption within the benchmarks of Peale et al. in order to measure only the overhead of their message franking scheme.

We show a comparison of communication overhead in Table 2.4, and we compare computation overhead in Table 2.5 for a message size of 1 kB. The benchmarks of TGLMR [238] were orders of magnitude slower than the other works because their construction of designated verifier signature performs more group operations; their communication overhead was also the highest. The comparison between **Hecate** and Peale et al. [202] is more nuanced. We stress that **Hecate** achieves additional security properties like anonymity and backward security. As a consequence, senders perform more work in **Hecate** and transmit more data, whereas Peale et al. leverage a non-anonymous network so that the platform can tag the originator of a message. On the other hand, Peale et al. require a forwarder to generate a commitment, whereas **Hecate** only requires generating a random 32 byte string (which could even be sampled beforehand).

	Hecate	No Hecate	Diff
Send	2.55 <i>ms</i>	2.38 <i>ms</i>	0.16 <i>ms</i>
Receive	3.19 <i>ms</i>	2.52 <i>ms</i>	0.67 <i>ms</i>
Total	5.74 <i>ms</i>	4.91 <i>ms</i>	0.83 <i>ms</i>
E2E Latency	37.28 <i>ms</i>	36.3 <i>ms</i>	0.98 <i>ms</i>

Table 2.6: Computation and communication costs for `signal-cli` with and without `Hecate`, for a message size of 1 kB. Send and Receive (10^4 trials) correspond to local computation prior to sending or after receiving the message over the network. E2E Latency (600 trials) starts at the beginning of Send and stops at the end of Receive, with network latency included.

2.7.2 End-to-End Prototype Deployment

In this section, we integrate `Hecate` into an existing Signal client and show that `Hecate` adds minimal overhead.

Implementation. To test the end-to-end overhead of sending and receiving messages, we integrated our Rust `Hecate` library into the Java tools `signal-cli` [220] and `libsignal-client` [224]. The sender’s `Frank` procedure adds the `Hecate` payload to a message before encrypting it using the EEMS, and then appends the `Hecate` envelope. The receiver decrypts the franked message and runs `Verify`. Our modified libraries are available as open source repositories [141, 142]. The sender gets tokens by running the Rust library prior to the start of the experiment.

We deployed the sender and receiver `signal-cli` instances on one machine with a 1.90GHz Intel i7-8650U CPU and 16GB of RAM running Ubuntu 20.04 LTS. They were connected over a wide-area network to an instance of `signal-server` [226].

Evaluation. We measured the client side overhead of running `signal-cli` with and without `Hecate` on messages of size 1 kB. In both cases, we measured the average of 10,000 trials of running local `signal-cli` operations and 600 trials of end-to-end

(E2E) latency, with outliers removed. Our timer for the end-to-end latency test starts as soon as the source’s `signal-cli` begins franking a message, and it ends when the receiver’s `signal-cli` completes processing the franked ciphertext and outputs the message. These sample sizes are larger than in §2.7.1 to overcome the noise added by the network latency, the polling rate of the receiver, and the warm-up time of the `jvm` instance of `signal-cli` for each of the parties.

Our results are shown in Table 2.6. They showcase how the findings from Table 2.5 translate to imperceptible overheads in an actual deployment of `Hecate` on a Signal client. The inclusion of `Hecate` adds less than a millisecond of runtime locally and over the network, on average. Moreover, this difference is dwarfed by the sample variance of `signal-cli` due to the sources of measurement uncertainty.

2.8 Conclusion and Discussion

In this work, we constructed the first abuse reporting protocol that combines asymmetric message franking and source tracing. We integrated this construction into a Signal client and showed that its performance impact was imperceptible. Along the way, we generalized the AMF model to accommodate preprocessing, and we formalized security properties that hadn’t previously been considered by message franking schemes.

In this final section, we discuss some extensions of `Hecate`, known limitations, and opportunities for future work.

2.8.1 Extensions

We extend `Hecate`’s communication from the two-device setting to more realistic flows supported by Signal.

Group Messaging. `Hecate`’s definitions and constructions can be ported in a straightforward manner to Signal’s group messaging protocol, in which broadcasts

to a group of size N are implemented via N individual point-to-point messages, after a server-assisted consensus protocol to determine the group [77]. We note that there exist recent works and an IETF standardization effort on sub-linear ends-to-ends encrypted group chats [28, 29, 52, 86, 218]; it remains an open problem to design abuse reporting mechanisms for these protocols. We do not yet know if there are any efficiency gains and added benefits to federating content moderation among different members of a group.

Multiple Devices. **Hecate** can easily support multiple devices for the same user (e.g., a phone and laptop) by giving each device its own independent set of tokens. The moderator can use the same id_{src} for both sets of tokens, so that reports only name an identity rather than a device.

2.8.2 Limitations

We discuss a few limits of our approach.

Reporting Benign Messages. Our construction allows receivers to report messages that may later be deemed to be non-abusive. While it might be possible to require the receiver to prove to an honest moderator that the message they are reporting is actually abusive, this question is incredibly delicate and is therefore out of scope for this and all prior works on end-to-end abuse reporting. In the special case that the receiver is colluding with the moderator, we remark that (a) there is little that can be done to prevent false reports, and (b) the overall leakage is no worse than what EEMSs would already reveal to this colluding set.

Distinguishing Forwarded vs. Original Messages. In our construction, receivers can distinguish between sent and forwarded messages. While this may be a desirable feature in a messaging app, it is still a leakage in our system.

Forwarding Cycle Linkability. If the forwarding path of a message contains a cycle, i.e. a receiver receives the same forwarded message multiple times, then they can tell

that these messages originate from the same source. This is an inherent weakness of **Hecate** as a result of forwarding the same tokens per message that we do not attempt to protect against. It may be possible to combine **Hecate** with Peale et al.’s techniques to remove this leakage [202].

Forwarding Tree Up-Rooting. Receiver’s of a message may “up-root” its forwarding sub-tree by acting as the original senders of that message instead of forwarding it themselves. In general, we do not believe that this poses a concern as users do not have the incentive to incriminate themselves with bad messages. Moreover, prior work [240] suggests that this problem warrants an application side solution, if any, and not a cryptographic one that would restrict users from copying received messages and acting as their creators.

2.8.3 Future Work

Looking ahead, we believe there exist at least five possible avenues of future research in the space of privacy-respecting content moderation.

Content Censorship. Content moderation systems can be misused for censorship purposes. Questions surrounding what constitutes misinformation or a “bad” message fall outside the scope of this work and into the realm of policy making and social media regulation. We believe however that it may be interesting to federate the role of the moderator in: (1) defining bad messages, (2) verifying reports, (3) taking actions with respect to flagged contents and users.

Super Spreaders. A recent line of work [227,254] on misinformation spread in social media distinguishes between honest users who forward misinformation and malicious actors that act as super spreaders of misinformation. Honest users can mistakenly forward or send misinformation content without ever realizing it. Super spreaders on the other are adversarially creating or spreading bad content. Future work could

examine aggregate behavior in order to distinguish malicious vs. mistaken users.

Partial opening. Known AMF constructions like **Hecate** only allow the receiver to report all or none of a message. It should be possible to achieve partial opening to the moderator by extending the message franking techniques of Leontiadis and Vaudenay [175] to the asymmetric setting.

Stronger Notions of Backward Security. Backward security makes no guarantees with respect to anything created during the time of compromise. In the context of content moderation, this implies that the adversary can blame users for old compromised messages. We encourage future research into ways to limit the damage of adversarial moderation reports or to allow honest parties to correct the record post-recovery.

Ensuring System Security. Finally, we emphasize that our study of abuse reporting has been primarily through a cryptographic lens, and as a result does not capture all aspects of security. For example, many of our crypto definitions assume that clients already have sufficient preprocessing tokens in hand. When implementing **Hecate**, careful attention is required to ensure that adversaries cannot obtain a side channel by, for example, influencing when preprocessing is run. We encourage cryptographers, systems security researchers, usability experts, and domain specialists to investigate whether and how to integrate **Hecate** (or any abuse reporting mechanism) into an end-to-end encrypted messaging system in a matter that promotes online trust, safety, and security.

Chapter 3

Batched Differentially Private Information Retrieval

This chapter is based on joint work [13] with Kinan Dak Albab, Mayank Varia and Kalman Graffi.

3.1 Introduction

Private Information Retrieval (PIR) [84, 165] is a cryptographic primitive that allows a client to retrieve a record from a public database held by a single or multiple servers without revealing the content of her query. PIR protocols have been developed for a variety of settings, including information theoretic PIR where the database is replicated across several servers [84], and computational PIR using single server [165]. The different settings of PIR are limited by various lower bounds on their computation or communication complexity. In essence, a server must “touch” every entry in the database when responding to a query, or else the server learns information about the query, namely what the query is not!

Recent PIR protocols [91, 162, 179, 200] achieve sub-linear computation and communication by relying on a preprocessing/offline stage that shifts the bulk of computation into off-peak hours [46], relaxing security to allow limited leakage [233], or batching queries, mostly in the case when they originate from the **same** client. These advances allowed PIR to be used in a variety of applications including private presence discovery [60, 199], anonymous communication and messaging [32, 80, 166, 189], private

media and advertisement consumption [129, 135], certificate transparency [179], and privacy preserving route recommendation [253].

Existing sublinear PIR protocols are able to handle medium to large databases of size n and still respond to queries reasonably quickly. However, they scale poorly as the number of queries increase: the sub-linear cost (e.g. \sqrt{n} for Checklist [162]) of handling each query quickly adds up when the number of queries approaches or exceeds the size of the database into a super-linear overall cost (e.g. $n\sqrt{n}$). Efficiently batching such queries and amortizing their overheads is an open problem when these queries are made by **different** clients: existing work that batches such queries assumes the number of queries is much smaller than the database size [179], burdens clients with making noise queries [233], or requires clients to closely coordinate and share secrets when preprocessing is used [46]. This complicates efforts to deploy PIR in a variety of important applications including software updates, contact tracing, content moderation, blacklisting of fake news, software vulnerability look-up, and similar large-scale automated services. We demonstrate this empirically in section 3.2.

In this work, we introduce DP-PIR, a novel differentially private PIR protocol tuned to efficiently handle large batches of queries approaching or exceeding the size of the underlying database. Our protocol batches queries from different non-coordinating clients. DP-PIR is the first protocol to achieve constant amortized server computation and communication, as well as constant client computation and communication.

While the details of our protocol are different from earlier work, at a high level our construction combines three ideas:

1. Offloading public key operations to an offline stage so that the online stage consists only of cheap operations [91, 200].
2. High throughput batched shuffling of messages by mixnets and secure messag-

ing systems [172, 173, 236, 243].

3. Relaxing the security of oblivious data structures and protocols to differentially private leakage [186].

DP-PIR Overview Our protocol is a batched multi-server PIR protocol optimized for queries approaching or exceeding the database size. DP-PIR is secure up to selective aborts against a dishonest majority of *malicious* servers, as long as at least one server is honest. Our protocol induces a per-batch overhead linear in the size of the database; this overhead is independent of the number of queries q in that batch, with a total computation complexity of $O(n + q)$ per entire batch. When the number of queries approaches or exceeds the size of the database, the amortized computation complexity per query is constant. Furthermore, our protocol only requires constant computation, communication, and storage on the client side, regardless of amortization. We describe the details of our construction in section 3.5.

Our protocol achieves this by relaxing the security guarantees of PIR to differential privacy (DP) [105]. Unlike traditional PIR protocols, servers in DP-PIR learn a noised differentially private histogram of the queries made in a batch. Clients secret share their queries and communicate them to the servers, which are organized in a chain similar to a mixnet. Our servers take turns shuffling these queries and injecting generated noise queries similar to Vuvuzela [243]. The last server reconstructs the queries (both real and noise) revealing a noisy histogram, and looks them up in the database. The servers similarly secret share and de-shuffle responses, while removing responses corresponding to their noise, and then send them to their respective clients for final reconstruction. The noise queries are generated from a particular distribution to ensure that the revealed histogram is (ϵ, δ) -differentially private, so that the smaller ϵ and δ get, the more noise queries need to be added. The distribution can be configured to provide privacy at the level of a single query or all queries made by the

same client in a single batch or over a period of time. The number of these noise queries is linear in n and $\frac{1}{\epsilon}$ and independent of the number of queries in a batch. The noise does not affect the accuracy or correctness of any client’s output. Section 3.3 describes our threat model and provides an interpretation of what this differentially oblivious [76] access pattern privacy means (as compared to traditional PIR).

Our protocol offloads all expensive public key operations to a similarly amortizable offline preprocessing stage. This stage produces correlated secret material that our protocol then uses online. Our online stage uses only a cheap information-theoretic secret sharing scheme, consisting solely of a few field operations, which modern CPUs can execute in a handful of cycles. The security of our protocol requires that this secret sharing scheme, which we define in section 3.4, is both incremental and non-malleable. Finally, section 3.6 describes how our protocol can be parallelized over additional machines to exhibit linear improvements in latency and throughput.

Our Contribution We make three main contributions:

1. We introduce a novel PIR protocol that achieves constant amortized server complexity with constant client computation and communication, including both its offline and online stage, when the number of queries is similar to or larger than the size of the database, even when the queries are made by different clients. Our offline stage performs public key operations linear in the database and queries size, and the online stage consists exclusively of cheap arithmetic operations.

2. We achieve a crypto-free online stage via a novel secret sharing scheme that is both incremental and non-malleable, based only on modular arithmetic for both sharing and reconstruction. To our knowledge, this is the first information theoretic scheme that exhibits both properties combined. This scheme may be of independent interest in scenarios involving Mixnets, (Distributed) ORAMs, and other shuffling and oblivious data structures.

3. We implement this protocol and demonstrate its performance and scaling to loads with hundreds of millions queries, while achieving throughput several fold higher than existing state of the art protocols. The experiments identify a criterion describing application settings where our protocol is most effective compared to existing protocols, based on the ratio of the number of queries over the database size.

3.2 Motivation

Private Information Retrieval is a powerful primitive that conceptually applies to a wide range of privacy preserving applications. Existing PIR protocols are well suited for applications with medium to large databases and small or infrequent number of queries [32,129,197,253]. However, they are impractical for a large class of applications with a large number of queries.

Motivating example One example that we consider throughout this work is checking for software updates on mobile app stores. The Google Play and iOS app stores contain an estimated 2.56 and 1.85 million applications each [145], and the number of active Android and iOS devices exceed 3 and 1.65 billion, respectively [92]. These devices perform periodic background checks to ensure that their installed applications are up to date. Currently, these checks are done without privacy: the app store knows all applications installed on a device, and can perform checks to determine if they are up-to-date quickly. However, the installed applications on one’s device constitute sensitive information. They can reveal information about the user’s activity (e.g. which bank they use), or whether the device has applications with known exploits.

It is desirable to hide the sensitive application information from the app store as well as potential attackers. A device can send a PIR query for each application installed, and the servers can privately respond with the most up-to-date version

label of each application. If the installed application is out of date, the device can then download the updated application via some anonymous channel, such as Tor. However, unlike DP-PIR, existing PIR protocols cannot scale to such loads, where the number of devices is about 1000x larger than the size of the database, each with tens of applications installed, given how quickly the sub-linear overheads per query add up. We demonstrate this empirically with three state of the art PIR protocols: Checklist [162], DPF [63], and SealPIR [31].

Additional Applications We believe that a large class of applications demonstrate similar properties ideal for DP-PIR. In privacy-preserving automated exposure notification for contact tracing [75, 234, 235], the number of recent cases in a city or region (i.e. the size of the database) is far smaller than the total population of that area (i.e. the number of queries). Similarly, identifying misinformation in end-to-end encrypted messaging systems [163] usually involves a denylist far smaller than the total number of messages exchanged in the system within a reasonable batching time window.

Our protocol relies on having two or more non-colluding parties that together constitute the service provider. This is a common assumption used by many other PIR protocols. Secure multiparty computation (MPC) has been applied in many real world applications over the last decade. This includes services federated over somewhat-independent subdivisions within the same large organization [20, 228], or additional parties that volunteer to participate to promote common social good [89, 212]. A third category, which we believe is most suited for the app store example, involves providers actively seeking out third parties to federate their services [55, 170] under contractual agreements for privacy or compliance reasons, usually in exchange for financial or reputation incentives. This has spurred various startups [192, 196] that provide their participation in secure multiparty computations as a service.

We believe that the differential privacy guarantees of DP-PIR suffice for applications where the primary focus is protecting the privacy of any given client, but not overall trends or patterns. Such as applications where it is also desirable for the (approximate) overall query distribution to be publicly revealed, e.g. an app store that displays download counts or a private exposure notification service that also identifies infection hotspots. DP-PIR is ideal for such applications, since it reveals a noised version of this distribution, without having to use an additional private heavy hitters protocol [56]. In practice, we emphasize that our relaxed DP guarantees should be viewed as an improvement over the insecure status-quo, rather than a replacement for PIR protocols that have stronger guarantees but impractical overheads in our target settings.

Comparison to Existing PIR Protocols Private Information Retrieval (PIR) has been extensively studied in a variety of settings. Information theoretic PIR replicates the database over several non-colluding servers [43], while computational PIR traditionally uses a single database and relies on cryptographic hardness assumptions [68, 82, 176].

Naive PIR protocols require a linear amount of computation and communication (e.g. sending the entire database over to the client), and several settings have close-to-linear lower bounds on either computation or communication [177].

Modern PIR protocols commonly introduce an offline preprocessing stage, which either encodes the database for faster online processing using replication [46, 57, 91, 162] and coding theory [64, 70, 137, 200], performs a linear amount of offline work per client to make the online stage sub-linear [70, 91, 162, 162], or performs expensive public key operations so that the online stage only consists of cheaper ones [91, 162, 200]. Other protocols rely on homomorphic primitives during online processing [23, 31, 248].

Finally, some protocols allow batching queries to amortize costs. When combined

Protocol	Computation		Communication	
	Online	Offline	Online	Offline
BIM04 [46]	$n^{0.55}$	–	$n^{0.55}$	–
CK20 [91]	\sqrt{n}	n	$\lambda^2 \log n$	\sqrt{n}
Checklist [162]	\sqrt{n}	n	$\lambda \log n$	\sqrt{n}
Naive †	n	–	n/q^*	–
PSIR [200] †	q^*n	n	$\log^c n$	n/q
CK20 [91] †	$q^*\sqrt{n}$	n	\sqrt{n}	\sqrt{n}/q^*
BIM04 [46] †§	$qn^{\frac{w}{3}}$	–	$n^{\frac{1}{3}}/q$	–
LG15 [179] †¶	$q^{0.8}n$	–	\sqrt{n}	–
This work †	$c_{\epsilon,\delta}n + q$	$c_{\epsilon,\delta}n + q$	1	1

†: support batching of queries made by the **same** client.
‡: supports batching of queries made by **different** clients.
§: amortizes to $n^{\frac{w}{3}}$, $w \geq 2$ is the matrix mult. exponent.
¶: up to $q = \sqrt{n}$.
||: amortizes to a constant when $q \sim n$.

Table 3.1: Computation and communication complexity of various existing PIR protocols. Here, n is the database size, q^* and q are the number of queries made by a single or different clients. For protocols that support batching, computation complexity represents the **total** complexity to handle a batch. Communication is always per query

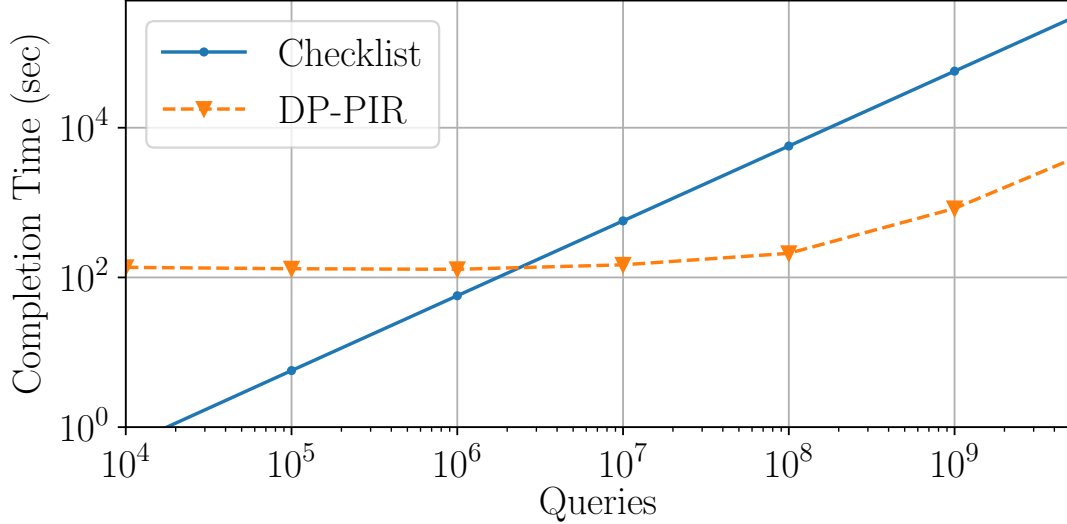


Figure 3.1: Checklist and DP-PIR Total completion time (y-axis, logscale) for varying number of queries (x-axis, logscale) against a 2.5M database

with preprocessing, batching is only supported for queries originating from the same client [91, 162, 200], or ones that share secret state [46]. Batching queries from different clients without preprocessing is possible [147] but has limitations. Earlier work induces a sublinear (but non constant) amortized computation complexity [46, 179]. Our work amortizes the computation costs of queries made by different queries down to a constant, while also requiring constant client work. In section 3.8, we discuss ϵ -PIR [233] which also amortizes such queries but burdens clients with generating the noise queries required for differential privacy.

Experiment Setup Our experiments measure the server(s) time needed to process a complete set of queries with $\epsilon = 0.1$ and $\delta = 10^{-6}$. While the trends shown in these results are intrinsic properties of our protocol design, the exact numbers depend on the setup and protocol parameters. Section 3.7 discusses our setup and the effects of these parameters in detail.

Checklist. Figure 3.1 shows the server computation time of Checklist and DP-

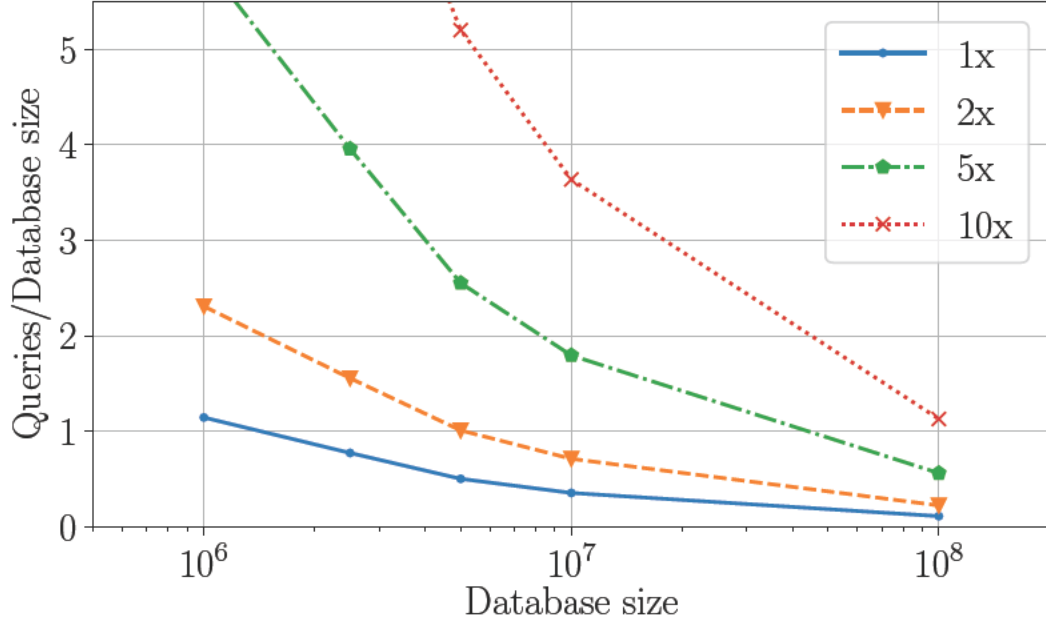


Figure 3.2: The ratios of queries/database (y-axis) after which DP-PIR outperforms Checklist by the indicated x factor for different database sizes (x-axis, logscale)

PIR when processing different number of queries against a database with $n = 2.5M$ elements. Our protocol has constant performance initially, which starts to increase with the number of queries q as they exceed 10M. In more detail, the computation time of DP-PIR is proportional to the total count of noise and real queries $c_{\epsilon, \delta}n + q$, where $c_{0.1, 10^{-6}} = 276$. Therefore, the cost induced by q is negligible compared to $c_{\epsilon, \delta}n$ until q becomes relatively significant.

On the other hand, Checklist scales linearly with the number of queries throughout, as its computation time is proportional to $q\sqrt{n}$. When the number of queries is small, this cost is far smaller than the initial overhead of our system. As q approaches n , both systems start getting comparable performance. DP-PIR achieves identical performance to Checklist at $q = 1.9M$ (slightly below $\frac{4}{5}$ the size of the database), and outperforms Checklist for more queries. Our speedup over Checklist grows with the ratio $\frac{q}{n}$, approaching a maximum speedup determined by \sqrt{n} when

the ratio approaches ∞ . For a database with 2.5M elements, our experiments demonstrate that we outperform Checklist by at least 2x, 5x, and 10x after the ratio exceeds 1.5, 3.9 and 8.1 respectively. We note that the largest data-point in the two figures are extrapolated.

The ratio required for achieving a particular speedup is not identical for all database sizes. As shown in Figure 3-2, DP-PIR prefers larger databases: the larger the database, the smaller the ratio required by DP-PIR to achieve a particular speedup, and the larger the maximum speedup that DP-PIR can achieve as $q \rightarrow \infty$.

We extrapolate from our empirical results to three possible scenarios for our Google Play store example, where the database contains roughly $2.5M$ elements with $3B$ active users, with the same setup and parameters as above. First, we assume each user makes exactly a single query (corresponding to a single app on their phone) resulting in a batch of size $q = 3B$, and $\frac{q}{n} = 1200$. In the second scenario, we assume each user checks the updates for all apps on their phone (e.g. say at most 100 apps), but only configure our system to provide DP guarantees only at the level of a single query (i.e. event-DP). In the last scenario, each user similarly makes 100 queries, but we configure our system to provide user-level DP guarantees protecting all the queries of the same user (i.e. user-DP), which results in adding 100 times the amount of noise. Our estimates indicate that our protocol will exhibit speedups of 161x, 180x, and 161x over checklist in these scenarios respectively. We discuss the different DP configurations in section 3.3.

The setup and parameters in both comparisons below is identical to what we've seen with Checklist.

DPF. Boyle, Gilboa, and Ishai [63] propose a PIR protocol based on distributed point functions (DPF). Unlike the offline-online protocol introduced in Checklist that uses punctured pseudorandom sets, DPF requires linear work in the database size to

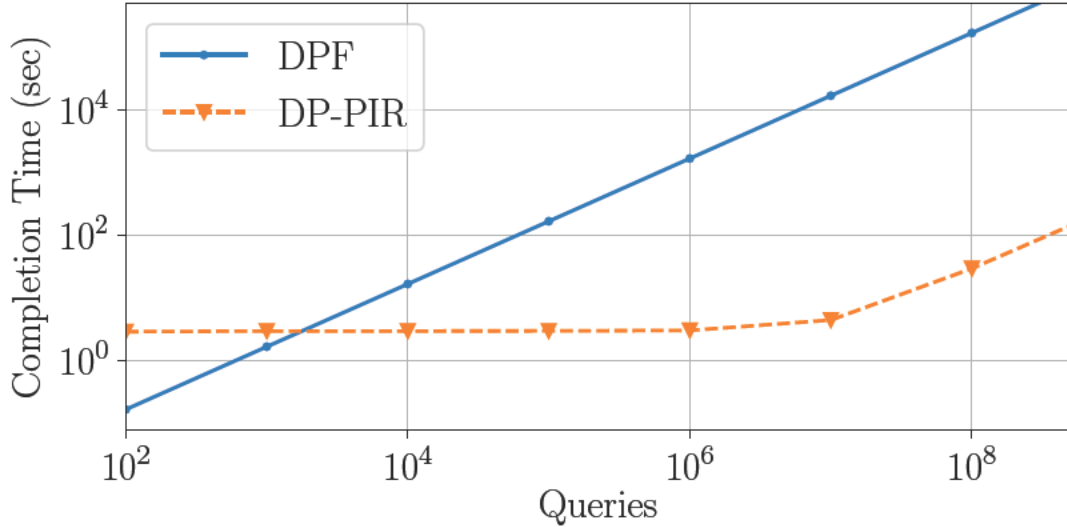


Figure 3-3: DPF and DP-PIR Total completion time (y-axis, logscale) for varying number of queries (x-axis, logscale) against a 2.5M database

handle user queries. However, DPF requires no offline preprocessing and significantly lower client computation and communication than checklist. We compare our system to the DPF implementation provided as an alternative backend for checklist based on the optimized implementation of Kales [152]. Figures 3-3 and 3-4 show our results. For a database with 100K elements, DPF outperforms DP-PIR when the number of queries is small relative to the size of the database. When the number of queries q approaches 31K, with $\frac{n}{q} = 0.0124$, the two systems exhibit identical completion time, with DP-PIR significantly outperforming DPF as the number of queries grow beyond that.

SealPIR. Figures 3-5 and 3-6 show similar results for SealPIR. In the first experiment, we use a database size of only 10K elements, and find that DP-PIR outperforms SealPIR at relatively few queries (around 32) with a ratio $\frac{q}{n}$ of just 0.003. Similarly, we achieve 2x, 5x, and 10x speedups for modest ratios all below 0.02. These ratios decrease as the database size grows, similar to our experiment with Checklist. We outperform SealPIR with far fewer queries than we do Checklist and DPF, in large part

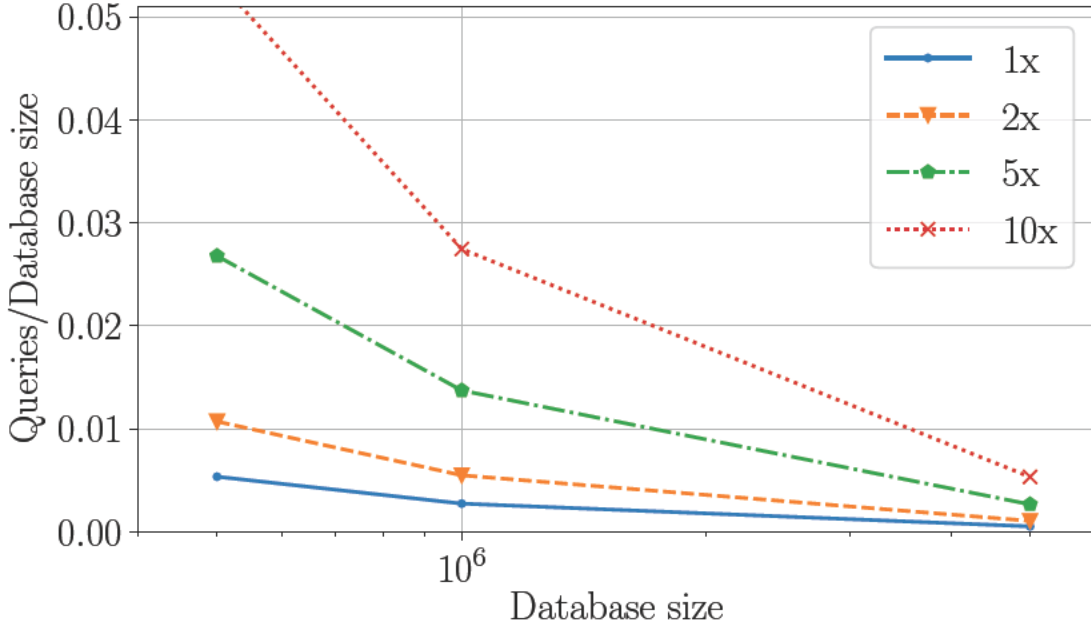


Figure 3-4: The ratios of queries/database (y-axis) after which DP-PIR outperforms DPF for different database sizes (x-axis, logscale)

because SealPIR’s uses expensive homomorphic operations during its online stage, while checklist offloads expensive linear work to an offline stage. Our protocol goes even further, only executing a couple of modular arithmetic operations per query online.

3.3 Protocol Overview

Our protocol consists of c_1, \dots, c_d clients and s_1, \dots, s_m servers. We designate s_1 and s_m as a special *frontend* and *backend* server respectively. We assume that every server s_i has a public encryption key pk_i known to all servers and clients, with associated secret key sk_i . Every server has a copy of the underlying database $T = K \rightarrow (V, \Sigma)$ mapping keys to values and signatures, such that $T[k] = (v, \sigma)$, where σ is a (m, m) -threshold signature over (k, v) by the m servers. The signatures are only needed for integrity and do not affect the privacy of clients; they allow clients to verify that the responses

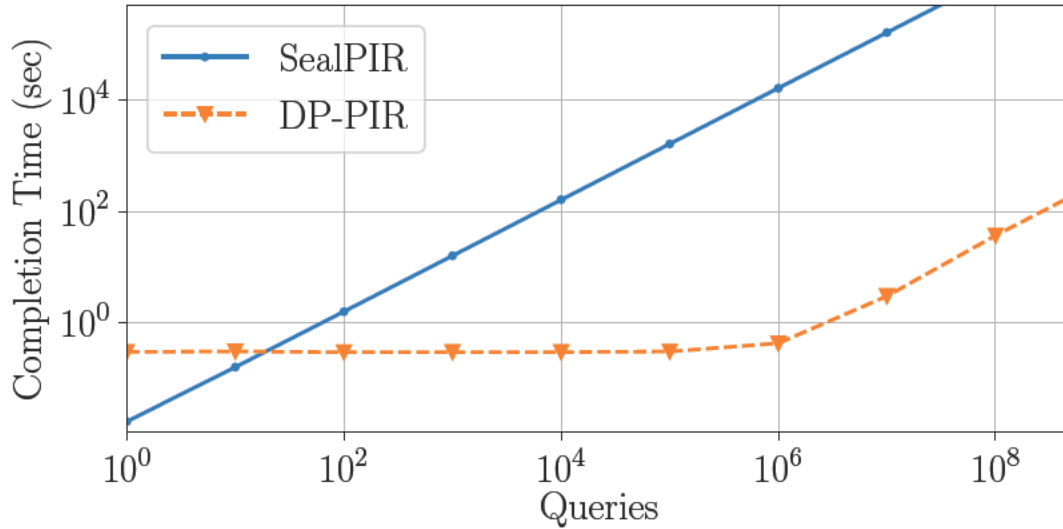


Figure 3.5: SealPIR and DP-PIR Total completion time (y-axis, logscale) for varying number of queries (x-axis, logscale) against a 10K database

they received correspond to the correct T agreed upon by the servers, and can be omitted when the backend is assumed to be semi-honest. We refer to the query made by client c_i by q^i , and its associated response by $r^i = (v^i, \text{Sig}^i)$.

3.3.1 Setting

Our protocol is easiest to understand in the case of a single epoch consisting of an input-independent offline stage followed by an online stage. The client state, created in the offline stage and consumed in the online one, consists exclusively of random elements. Clients can store the seed used to produce these elements to achieve constant storage relative to the number of queries and number of servers. A client need only submit her secrets to the service during the offline stage, and can immediately leave the protocol afterwards. The client can reconnect at any later time to make a query without any further coordination.

The offline stage is more computationally expensive than the online one, since it performs a linear number of public key operations overall. We suggest that the

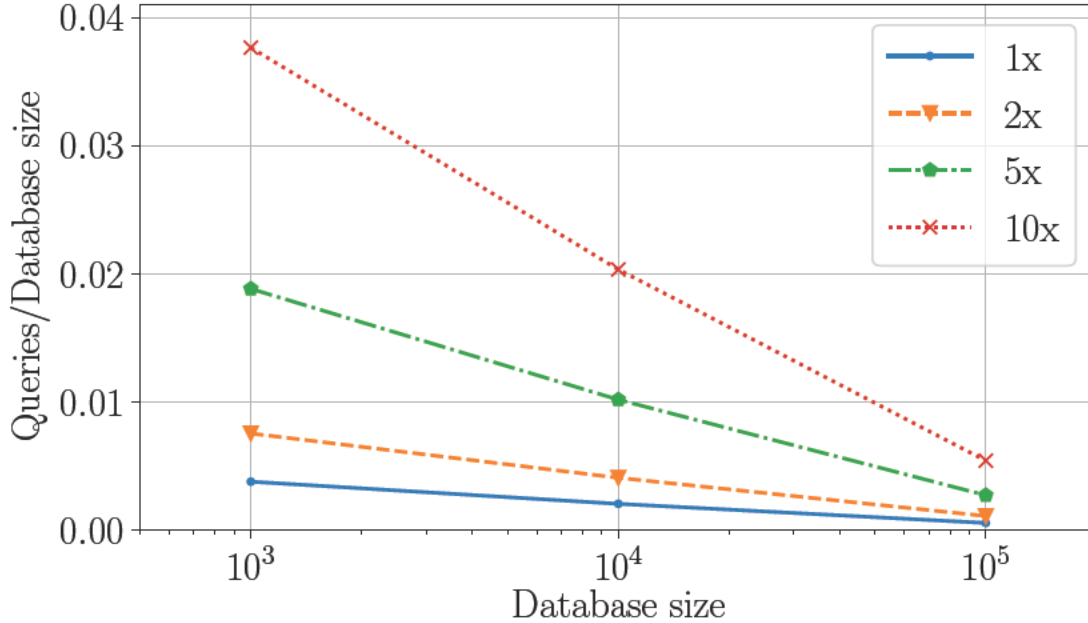


Figure 3-6: The ratios of queries/database (y-axis) after which DP-PIR outperforms SealPIR for different database sizes (x-axis, logscale)

offline stage be carried out during off-peak hours (e.g. overnight), when utilization is low. Furthermore, both our stages are embarrassingly parallel in the resources of each party. It may be reasonable to run the offline stage with more resources, if these resources are cheaper to acquire overnight (e.g. spot instances). Our offline stage is similar to Vuvuzela [243], which exhibits good throughput. However, the linear number of public key operations performed by Vuvuzela makes it impractical for our online stage. Indeed, our online stage is crypto-free using only a handful of arithmetic operations per query.

In practice, services using DP-PIR alternate between collecting a batch of queries submitted from clients within some configurable time window, and processing that batch using our online protocol. In section 3.7, we discuss the effects of this batching window on our performance. Each batch requires corresponding offline processing. Our protocol allows multiple offline stages (e.g. the ones corresponding to an entire day’s worth of batches) to be pooled together into a proportionally larger stage ex-

ecuted in one shot during off-peak hours when resources are cheaper (e.g. the night before). The clients can choose to make their queries at any time after preprocessing, but client states from several uncombined offline stages should not be used in a single online batch, to avoid allowing the adversary to identify the origin of the query by diffing out clients that participated in different stages.

Our protocol assumes that T and its signatures are provided as input. Thus, the servers must agree on T and produce signatures for it ahead of time. The same T and signatures can be reused by many offline/online stages; servers need only compute new signatures when the underlying database changes, and may rely on timestamps to enable clients to reject expired responses. The servers never sign or verify any signatures during either the offline or online stages, and each client needs to verify one signature per received response. Therefore, the efficiency of signing/verification is secondary. Instead, our protocol prefers signature schemes that produce shorter signatures for lower bandwidth.

3.3.2 Threat Model

Our construction operates in the ‘anytrust’ model up to *selective* abort. Specifically, we tolerate up to $m - 1$ malicious servers and $d - 1$ malicious clients.

In terms of *confidentiality*, our protocol differs from traditional perfectly-private PIR protocols in that it leaks noisy access patterns over the honest clients’ queries, in the form of a differentially private noisy histogram $\mathcal{H}(Q) = H_{\text{honest}}(Q) + \chi(\epsilon, \delta, \phi)$.

As for *integrity*, our protocol is secure up to *selective* abort, and does not guarantee fairness. Adversarial servers may elect to stop responding to queries, effectively aborting the entire protocol. Furthermore, they can do so selectively: any server can decide to drop queries at random, the frontend server can drop queries based on the identity of their clients, and the backend server can drop queries based on their value.

We stress that an adversary cannot drop a query based on the conjunction of the

client’s identity and the value, regardless of which subset of servers gets corrupted. Also, an adversary can only drop a query, but cannot convince a client to accept an incorrect response, since clients can validate the correctness of received responses locally.

3.3.3 Interpreting Privacy

Our protocol can be configured to provide different levels of (ϵ, δ) -differential privacy by selecting the parameters of the underlying distribution used to sample noise queries. The most efficient (and easiest to understand) configuration is often called *event-DP*, which provides guarantees at the level of any **single** honest query. Another DP configuration, commonly termed *user-DP*, provides guarantees at the level of **all** queries made by any honest client. We use event-DP throughout the paper except when otherwise noted.

We provide either guarantee at the level of a single isolated batch. In particular, we consider two batches of queries Q and Q' over the honest clients’ queries to be ϕ -neighboring batches when they consists of identical queries except for ϕ queries. In event-DP, it is enough to consider $\phi = 1$. While in user-DP, we set ϕ to the number of queries a client can make within a batch (or an upper bound of it). In either case, the sensitivity is 2ϕ , which means that for the same ϵ, δ the expected number of noise queries we add grows linearly in ϕ .

Definition 12 (Differentially Private PIR Access Patterns). *For any privacy parameters ϵ, δ , and every two ϕ -neighboring batches of queries Q, Q' , the probabilities of our protocol producing identical access pattern histograms are (ϵ, δ) -similar when run on either set:*

$$Pr[\mathcal{H}(Q) = H] \leq e^\epsilon Pr[\mathcal{H}(Q') = H] + \delta$$

Our definition uses the substitution formulation of DP, rather than the more

common addition/removal; see [242, §1.6] for details. Substitution is commonly used in secure computation protocols involving DP leakage [186]. We use this variant since our protocol does not hide whether a client made a query in a batch or not: the adversary already knows this e.g. by observing IP addresses associated to queries. Instead, we hide the value of the query itself. Substitution is more conservative adding twice the expected amount of noise queries, since its sensitivity is 2ϕ compared to ϕ in the other.

So far, we only discussed guarantees within a single online stage. In any long running DP system where clients can make unbounded queries, it is impossible to achieve user-DP globally. Instead, practical systems [180] often rely on the user-time-DP model, where the guarantees extend over all queries made by a client over a set moving time window (e.g. a week). We can achieve this by setting ϕ to the number of queries that a client may make over a time window, regardless of how the client distributes the queries over the batches in that window. This follows from DP’s composition theorem.

One way to interpret our DP guarantees (aka “differential obliviousness” [76]) is that they provide any client with plausible deniability: a client that made queries q_1, \dots, q_ϕ over some period of time can claim that her true queries were any different q'_1, \dots, q'_ϕ , and external distinguishers cannot falsify this claim since the probability of either case inducing any same observed histogram of access patterns is similar.

Whereas traditional differential privacy mechanisms trade privacy for accuracy, differential obliviousness trades privacy for performance while always providing accurate outputs. In DP-PIR, increasing privacy (by lowering ϵ and δ or increasing ϕ) results in additional noise queries, making our protocol proportionally slower, and requiring a proportionally larger batch of queries to achieve the same amortization, and thus speedup, over other protocols. The amount of noise queries scales linearly

in ϕ and $\frac{1}{\epsilon}$ and sub-linearly in δ (see Table 3.3).

3.4 Incremental Non-Malleable Secret Sharing

Our protocol relies on shuffling real queries with noise queries by our chain of servers, similar to Vuvuzela and other mixnets where public key onion encryption is used to pass secrets through that chain. However, this induces a large number of public key operations, proportional to $m \times |batch|$. We use a novel cheaper arithmetic-based secret sharing scheme instead of onion encryption during our online stage.

The secret sharing scheme provides similar security guarantees to onion encryption, to ensure that input and output queries are untraceable by external adversaries:

1. *Secrecy*: As long as one of the shares is unknown, reconstruction cannot be carried out by an adversary.

2. *Incremental Reconstruction*: A server that only knows a single secret share and a running tally must be able to combine them to produce a new tally. The new tally must produce the original secret when combined with the remaining shares.

3. *Independence*: An adversary cannot link any partially reconstructed output from a set of outputs to any shared input in the corresponding input set.

4. *Non-Malleability*: An adversary who perturbs any given share cannot guarantee that the output of reconstruction with that perturbed share satisfies any desired relationship. In particular, the adversary cannot perturb shares such that reconstruction yields a specific value (e.g., 0), or a specific function of the original secret (e.g., adding a fixed offset).

Formally, we define a secret sharing scheme with incremental reconstruction with the usual sharing mechanism but a new method to recover the original secret.

Definition 13. *An incremental secret sharing scheme S over a field \mathbb{F} and m parties contains two algorithms.*

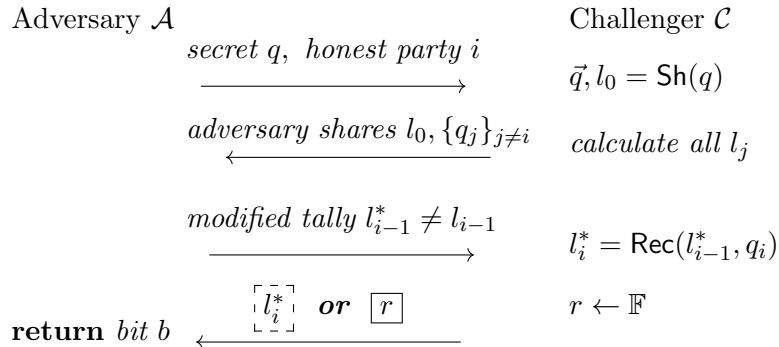
- $\text{Sh}(q)$ disperses a secret q into a randomly chosen set of shares $\vec{q} = q_1, \dots, q_m \in \mathbb{F}$ and some initial tally l_0 .
- $\text{Rec}(l_{i-1}, q_i) \rightarrow l_i$ performs party i 's partial reconstruction to produce running tally l_i .

The scheme is correct if for all sharings $(\vec{q}, l_0) \leftarrow \text{Sh}(q)$, the overall reconstruction returns $l_m = q$.

Non-malleability is critical for preserving security when the last (backend) server is corrupted. The backend can observe the final reconstructed values of all queries to identify queries perturbed by earlier colluding servers. If the perturbation can be undone (e.g. by removing a fixed offset), then the backend can learn the value of the query and link it to information known by other servers, such as the identity of its client.

We formally define non-malleability through the following indistinguishability game. It guarantees that if an adversarial set of $m - 1$ parties submits a tampered partial tally l_{i-1}^* to the honest party i , then the tally l_i^* returned by the honest party is uniformly random. As a result, l_i^* is independent of (and therefore hides) the secret q , and it only completes to a reconstruction of q with probability $1/|\mathbb{F}|$.

Definition 14. Consider the following two games that only differ in the final step. Call them $\boxed{\text{Left}}$ and $\boxed{\text{Right}}$ respectively.



We say that an **incremental** secret sharing scheme $S = (\text{Sh}, \text{Rec})$ is **non-malleable** if for all adversaries \mathcal{A} , the Left and Right games are (perfectly) indistinguishable.

Several non-malleable secret sharing schemes exist [37,127]. However, they are not incremental: their reconstruction is a one-shot operation over all shares. Conversely, known incremental schemes, such as additive or XOR-based sharing, are vulnerable to malleability. It would have been possible to use different primitives in our protocol that satisfy our desired properties, such as authenticated onion symmetric-key encryption. However, these operations remain more expensive than simple information theoretic secret sharing schemes that can be implemented with a handful of arithmetic operations.

Our Incremental Sharing Construction Given a secret q , a prime modulus z , and an integer m , our scheme produces $m+1$ pairs $q_0 = (x_0, y_0), q_1 = (x_1, y_1), \dots, q_m = (x_m, y_m)$, where each pair represents a single share of q . All x and y values are chosen independently at random from \mathbb{F}_z and \mathbb{F}_z^* respectively, except for the very first pair x_0, y_0 , whose values are set to:

$$x_0 = \langle [(q - x_m) \times y_m^{-1}] \dots - x_1 \rangle \times y_1^{-1} \text{ mod } z, \quad y_0 = 0.$$

All shares except the first one can be selected prior to knowing q . This is important for our offline stage. The modulus z must be as big as the key size in the underlying database (32 bits in our experiments). To reconstruct the secret q , we show below the incremental reconstruction operations $\text{Rec}(l_{i-1}, q_i)$ to construct the first partial tally and all subsequent ones:

$$l_0 = y_0 \times 1 + x_0 \text{ mod } z,$$

$$l_j = y_j \times l_{j-1} + x_j \text{ mod } z.$$

Correctness (i.e., $l_m = q$) stems from our choice of (x_0, y_0) .

Theorem 3.4.1 (Non-malleability of our Incremental Secret Sharing Scheme). *Let $S = (\text{Sh}, \text{Rec})$ denote our incremental secret sharing construction from §3.4. This scheme satisfies the non-malleable property of Def. 14.*

Proof. Let \mathcal{A} be an adversary who plays the non-malleability game. Suppose that it chooses a secret q and honest party i in the first round of the game. The adversary receives back in response the shares $\vec{q}' = \{q_j\}_{j \neq i}$ and the initial tally l_0 , from which it can compute all legitimate partial tallies including l_{i-1} and l_i .

Our aim is to show that for any modified partial tally $l_{i-1}^* \neq l_{i-1}$ that the adversary might choose in the second round of the game, the resulting partial tally after the honest party $l_i^* \equiv r$ is (perfectly) indistinguishable from random even given the adversary's knowledge of q and \vec{q}' . In the Right game, the random value $r \in \{0, 1, \dots, z-1\}$ is uniformly sampled from the space of all elements in \mathbb{F}_z . Ergo, it suffices to show that any value of l_i^* within $\{0, 1, \dots, z-1\}$ is equally probable in the Left game, where this probability is taken over the challenger's sampling of the honest party's share (i.e., the one piece of randomness that is hidden from the adversary).

Fix an arbitrary choice of l_{i-1}^* and l_i^* , subject to the game's requirement that $l_{i-1}^* \neq l_{i-1}$. Let $q_i = (x_i, y_i)$ denote the honest party's share, which the adversary \mathcal{A} does not know. We observe that there exists exactly one choice of q_i that is consistent with both (i) the secret q and associated partial tallies l_{i-1} and l_i and (ii) returning l_i^* in the Left game. These constraints impose the following system of two linear

equations in two unknowns:

$$x_i + y_i \times l_{i-1} = l_i \text{ mod } z$$

$$x_i + y_i \times l_{i-1}^* = l_i^* \text{ mod } z$$

Since $l_{i-1} \neq l_{i-1}^*$ and z is prime (meaning that nonzero entries are invertible mod z), this system of equations has exactly one solution:

$$x_i = l_i - l_{i-1} \times (l_i - l_i^*) \times (l_{i-1} - l_{i-1}^*)^{-1} \text{ mod } z$$

$$y_i = (l_i - l_i^*) \times (l_{i-1} - l_{i-1}^*)^{-1} \text{ mod } z$$

Therefore, even with l_{i-1}^* chosen by the adversary, any execution of the Left game results in l_i^* having the uniform distribution conditioned on \mathcal{A} 's view, so it is perfectly indistinguishable from the Right game as desired. \square

3.5 Our DP-PIR Protocol

Offline Stage Our offline stage consists of a single sequential pass over the m servers. Clients generate random secrets locally, and submit them after onion encryption to the first server in the chain. The first server receives all such incoming messages from clients, until a configurable granularity is reached, e.g. after a certain time window passes or a number of messages is received. All incoming messages at that point constitutes the input set for that server. The server outputs a larger set. This set

Algorithm 1 Client i Offline Stage

Input: Nothing.

Output state at the client: a list of anonymous secrets $[a_0^i, \dots, a_m^i]$, one per each of the m servers. The client uses these secrets in the online stage.

Output to s_1 : Onion encryption of a_1^i, \dots, a_m^i .

1. **Generate Random Values:** For each Server s_j , the client generates 4 values all sampled uniformly at random: (1) A globally unique identifier t_j^i . (2) Two incremental pre-shares $x_j^i \in \mathbb{Z}_z$ and $y_j^i \in \mathbb{Z}_z^*$. (3) An additive pre-share $e_j^i \in [0, 2^b)$. We define $e^i = \sum e_j^i \pmod{2^b}$ which the client uses to reconstruct the response online.
 2. **Build Shared Anonymous secrets:** The client builds $a_j^i = (t_j^i, t_{j+1}^i, x_j^i, y_j^i, e_j^i)$, for every server $1 \leq j \leq m$, using the generated random values above, with $t_{m+1}^i = \perp$. These secrets are stored by the client for later use in the online stage.
 3. **Onion Encryption:** The client onion encrypts the secrets using the correspond server's public key, such that $OEnc_m^i = Enc(\mathbf{sk}_m, a_m^i)$ and $OEnc_j^i = Enc(\mathbf{sk}_j, a_j^i \parallel OEnc_{j+1}^i)$.
 4. **Secrets Submission:** The client sends the onion cipher $OEnc_1^i$ to server s_1 . The client can leave the protocol as soon as receipt of this message is acknowledged.
-

Algorithm 2 Server s_j Offline Stage

Configuration: The underlying database $T : K \rightarrow (V, \Sigma)$, and privacy parameters ϵ, δ, ϕ .

Input from s_{j-1} or clients if $j = 1$: A set of onion ciphers of anonymous secrets, one per each incoming request.

Output state at s_j : A mapping M of unique tag t_j^i to its corresponding shared anonymous secrets a_j^i used to handle incoming queries during the online stage. A list of generated anonymous secrets L used to create noise queries during the online stage. A sampled histogram \mathcal{N} of noise queries to use in the online stage.

Output to server s_{j+1} : A set output onion ciphers corresponding to input onion ciphers and noise generated by s_j .

1. **Onion Decryption:** For every received onion cipher $OEnc_j^i$, the server decrypts the cipher with its secret key sk_j , producing a_j^i and $OEnc_{j+1}^i$.
 2. **Anonymous Secret Installation:** For every decrypted secret $a_j^i = (t_j^i, t_{j+1}^i, x_j^i, y_j^i, e_j^i)$, the server stores entry $(t_{j+1}^i, x_j^i, y_j^i, e_j^i)$ at $M[t_j^i]$ for later use in the online stage.
-

contains both the processed input messages, as well as new messages inserted by the server.

The client-side protocol is shown in algorithm 1. Concretely, for each server j , client i generates secret $a_j^i = (t_j^i, t_{j+1}^i, x_j^i, y_j^i, e_j^i)$, where t_j^i and t_{j+1}^i are random tags chosen from a sufficiently large domain that the client uses online to point each server to its secret without revealing its identity, x_j^i and y_j^i are secret shares from our

Algorithm 2 Continued: Server s_j Offline Stage

If $j < m$:

5. **Noise Pre-Sampling:** The server samples a histogram representing counts of noisy queries to add for every key in the database $\mathcal{N} \leftarrow \chi(\epsilon, \delta, \phi)$, and computes the total count of this noise $S = \sum \mathcal{N}$.
 6. **Build shared anonymous secrets for noise:** The server generates S many anonymous secrets and onion encrypts them for all $s_{j'}$ with $j' > j$, using the same algorithm as the client. The server stores these secrets in L .
 7. **Shuffling and Forwarding:** The server shuffles all onion ciphers, including all $OEnc_{j+1}^i$ decrypted in step (1) or generated by step (4), and sends them over to the next server s_{j+1} .
-

incremental secret sharing scheme used to reconstruct the query, and $\sum e_j^i \bmod 2^b = e^i$ are additive secret shares used to mask the response. The exponent b corresponds to the bit size of values and signatures (instantiated to $32 + 384$ in our experiments). Our offline protocol uses *onion encryption* from CCA-secure public key encryption to pass secrets through the servers (here, $::$ denotes string concatenation):

$$OEnc_1^i = Enc(\text{sk}_1, a_1^i :: Enc(\text{sk}_2, a_2^i :: \dots Enc(\text{sk}_m, a_m^i) \dots))$$

In addition to secrets from clients, each server must inject sufficiently many secrets at subsequent servers to handle all noise queries that the server needs to make in the online stage. This corresponds to steps 3 and 4 in algorithm 2, where the server computes the exact noise amount by pre-sampling.

The output set of each server contains onion ciphers, encrypted under the keys of the subsequent servers in the chain. None of the plaintexts decrypted by the current

Algorithm 3 Client i Online Stage

Input: A query q^i .

Input state at the client: a shared anonymous secrets $a_j^i = (t_j^i, t_{j+1}^i, x_j^i, y_j^i, e_j^i)$ per server s_j generated by the offline stage.

Output: A value v^i corresponding to $T[q^i]$.

1. **Compute Final Incremental Secret Share:** Client computes $l_1^i = x_0^i$, so that $(x_0^i, 0)$ combined with $(x_1^i, y_1^i), \dots, (x_m^i, y_m^i)$ is a valid sharing of q^i , per our incremental secret sharing scheme.
 2. **Query Submission:** Client sends (t_1^i, l_1^i) to server s_1 .
 3. **Response Reconstruction:** Client receives response r_1^i from s_1 and reconstructs $(v^i, \text{Sig}i) = r_1^i - e^i \pmod{2^b}$.
 4. **Response verification:** The client ensures that $\text{Sig}i$ is a valid signature over (q^i, r^i) by s_1, \dots, s_{m-1} .
-

server survives, they are all consumed and stored in the server's local mapping for use during the online stage. No linkage between messages in the input and output sets is possible without knowing the server's secret key, since the ciphers in the input cannot be used to distinguish between (sub-components of) their plaintexts, and since the output set is uniformly shuffled. This is true even if the adversary perturbs onion ciphers prior to passing them to an honest party (by CCA security), which in-essence denies service to the corresponding query.

Online Stage The client-side online protocol is shown in algorithm 3. The server-side online stage (algorithm 4) is structured similarly to the offline stage. However,

Algorithm 4 Server s_j Online Stage

State at s_j : The mapping M , list L , and noise histogram \mathcal{N} stored from the offline stage.

Input from s_{j-1} or clients if $j = 1$: A list of queries (t_j^i, l_j^i) .

Output to s_{j-1} or clients: A list of responses r_j^i corresponding to each query i .

1. **Anonymous Secret Lookup:** For every received query (t_j^i, l_j^i) , the server finds

$$M[t_j^i] = (t_{j+1}^i, x_j^i, y_j^i, e_j^i).$$

2. **Query Handling:** For every received query, the server computes output query

$(t_{j+1}^i, \text{Rec}(l_j^i, (x_j^i, y_j^i)))$, where Rec is our scheme's incremental reconstruction function.

If $j < m$:

3. **Noise injection:** The server makes output queries per stored noise histogram \mathcal{N} , using the stored list of anonymous secrets L and the client's online protocol. By construction, there are exactly as many secrets in L as overall queries in \mathcal{N} .
 4. **Shuffling and Forwarding:** The server shuffles all output queries, both real and noise, and sends them over to the next server s_{j+1} . The server waits until she receives the corresponding responses from s_{j+1} , and de-shuffles them using the inverse permutation.
-

it requires going through the chain of servers twice. The first phase (steps 1-4) moves from the clients to the backend server, where every server incrementally reconstructs

Algorithm 4 Continued: Server s_j Online Stage

If $j < m$:

5. **Response Handling:** Received responses corresponding to noise queries generated by this server are discarded. For every remaining received response r_{j+1}^i , the server computes the output response $r_j^i = r_{j+1}^i + e_j^i \bmod 2^b$.
6. **Response Forwarding:** The server sends all output responses r_j^i to s_{j-1} , or the corresponding client c_i if $j = 1$.

If $j = m$:

7. **Response Lookup:** The backend server does not need to inject any noise or shuffle. By construction, step (2) computes (\perp, q^i) for each received query. The backend finds the corresponding $T[q^i] = (v^i, \mathbf{Sig}i)$. If q^i was not found in the database (because a malicious party mishandled it), we return an arbitrary random value.
 8. **Response Handling:** The backend computes responses $r_j^i = (v^i \mathbin{::} \mathbf{Sig}i) + e_j^i \bmod 2^b$, and sends them to s_{j-1} .
-

the values of received queries using the stored secrets (steps 1-2), and injects its noise queries into the running set of queries (step 3). The second phase moves in the opposite direction (steps 5-6), with every server removing responses to their noise queries, and incrementally reconstructing the received responses, until the final value of a response is reconstructed by its corresponding client. The backend operates differently than the rest of the servers (steps 7-8). It computes the reconstructed

query set, and finds their corresponding responses in the database via direct look-ups. The backend need not add any noise queries, which alleviates the need for shuffling at the backend.

Discussion The security of both offline and online stages rely on the same intuition. First, an adversary that observes the input and output sets of an honest server should not be able to link any output message to its input. Second, the adversary must not be able to distinguish outputs corresponding to real queries from noise injected by that server.

The honest server shuffles and re-randomizes all its input messages, which guarantees that the adversary cannot link input and output messages. In the offline stage this re-randomization is performed with onion-decryption, while the online stage performs it using our non-malleable incremental reconstruction and additive secret sharing for its two phases respectively. We do not need to use a non-malleable secret sharing scheme for response handling, since the adversary cannot observe the final response output, which is only revealed to the corresponding client, and thus cannot observe the effects of a perturbation.

Shuffling in the noise with the re-randomized messages ensure that they are indistinguishable. A consequence of this is that a server cannot send out any output message until it receives the entirety of its input set from the previous server to avoid leaking information about the permutation used. Idle servers further along the chain can use this time to perform input independent components of the protocol, such as sampling the noise, building and encrypting their anonymous secrets, or sampling a shuffling order.

A malicious server may deviate from this protocol in a variety of ways: it may de-shuffle responses incorrectly (by using a different order), attach a different tag to a query than the one the offline stage dictates, or set the output value corresponding to

a query or response arbitrarily (including via the use of an incorrect pre-share). The offline stage does not provide a malicious server with additional deviation capability: any deviation in the offline stage can be reformulated as a deviation in the online stage, after carrying out the offline stage honestly, with both deviations achieving identical effects. Finally, a backend server may choose to provide incorrect responses to queries by ignoring the underlying database.

Each of these deviations has the same effect: the non-malleability of **both** our sharing scheme and onion encryption ensures that mishandled messages reconstruct to random values, and mishandled responses will not pass client-side verification unless the adversary can forge signatures. In either case, the affected clients will identify that the output they received is incorrect and reject it. Ergo, servers can only use this approach to selectively deny service to some clients or queries. A malicious frontend can deny service to any desired subset of clients since it knows which queries correspond to which clients, a malicious backend can deny service to any number of client who queried a particular entry in the database, and any server can deny service to random clients. The backend and frontend capabilities cannot be combined even when colluding since at least one honest server exists between the frontend and backend. These guarantees are similar to those of Vuvuzela [243] and many other mixnet systems.

Formal Security We rigorously specify our security guarantee in Theorem 3.5.1, which refers to the ideal functionality defined in Algorithm 5. The ideal functionality formalizes our notion of “selective” abort. In particular, it formalizes capabilities of the adversary to deny service to a specific query based on at most one of its value or its origin client.

Theorem 3.5.1 (Security of our protocol Π). *For any set A of adversarial colluding servers and clients, including no more than $m - 1$ servers, there exists a simulator S ,*

Algorithm 5 Ideal Functionality \mathcal{F}

Input: A set of queries q^i , one per client, the underlying database $T : K \rightarrow (V, \Sigma)$, and privacy parameters ϵ, δ, ϕ .

Output: A set of outputs v^i , one per client, either equal to the correct value or \perp .

Leakage: A noisy histogram \mathcal{H} revealed to s_m .

1. if s_1 is corrupted, \mathcal{F} receives a list of client identities from the adversary. These clients are excluded from the next steps, and receive \perp outputs.
 2. \mathcal{F} reveals the noised histogram $\mathcal{H} = H_{\text{honest}} + \mathcal{N}$ to the backend server s_m , where H_{honest} is the histogram of queries made by *honest* clients not excluded by the previous step, and \mathcal{N} is sampled at random from the distribution of noise $\chi(\epsilon, \delta, \phi)$.
 3. if the backend is corrupted, \mathcal{F} receives a list of counts c_i for every entry in the database k_i , and outputs \perp to c_i -many clients, randomly chosen among the remaining clients that queried k_i .
 4. if any server, other than s_m and s_1 , is corrupted, \mathcal{F} receives a number c , and outputs \perp to c -many clients, randomly chosen among the remaining clients.
 5. if s_1 is corrupted, \mathcal{F} receives an additional list of client identities to receive \perp .
 6. \mathcal{F} outputs v^i such that $T[q^i] = (v^i, \text{Sig}i)$ for every client i not excluded by any of the steps above.
-

such that for client inputs q^1, \dots, q^d , we have:

$$\text{View}_{\text{Real}}(\Pi, A, (q^1, \dots, q^d)) \approx \text{View}_{\text{Ideal}}(\mathcal{F}, \mathcal{S}, (q^1, \dots, q^d))$$

Input: $T : K \rightarrow (V, \Sigma)$, and ϵ, δ .

Simulating the Offline Stage: The offline stage has no inputs on the client side, and only needs access to T, ϵ , and δ on the server side. The simulator can simulate this stage perfectly by running our protocol when simulating honest parties, and invoking the adversary for corrupted ones.

Simulating Client Online Queries: The simulator uses “junk” queries for this simulation. The actual queries are injected by the simulator later during the query phase.

1. The simulator assigns random query values to each honest client in its head. The simulator then runs our client protocol for these input query values, providing each client with the anonymous secrets the simulator selected when simulating that client’s offline phase.
2. The simulator runs the adversary’s code to determine the query message of each corrupted client.

Simulating Server Online Protocol - First Pass: The simulator goes through the servers in order, from s_1 to s_{m-1} .

1. **If s_i is corrupted:** The simulator runs the adversary on the query vector constructed by the previous step, which outputs the next query vector.
2. **If s_i is the first non-corrupted server:**
 - **Neither s_1 nor the backend are corrupted:** The simulator executes step 3 below.
 - **If s_1 is corrupted:** The simulator begins by identifying any mishandled honest client queries in the current query vector. For each honest client query, the simulator looks for it by its tag, which the simulator knows because she simulated the offline stage of that client. The simulator validates that the associated tally has the expected value, furthermore, it

checks that the anonymous secret installed at s_i during the offline stage match the ones the simulator generated when simulating the client portion of that offline stage. All of these checks depend on the honest client and honest server s_i offline state, which the simulator knows.

If any of tags, tallies, or shares do not match their expected value, or are missing, then the simulator knows that the adversary has mishandled this client's query (or corresponding offline stage) prior to server s_i . The simulator sends the identities of all such clients to the ideal functionality (step 1 in \mathcal{F}).

- **If backend is corrupted:** The simulator receives a noised histogram H_{honest} from the ideal functionality. The simulator identifies all honest queries that have not been mishandled so far. Say there are k such queries. As part of simulating s_i , the simulator will replace the tallies of these queries with new tallies, such that the tally of honest query $w \leq k$ would reconstruct to the value of the w -th entry in H_{honest} , when combined with the remaining shares that the simulator generated for that client during its offline stage.

Furthermore, the simulator needs to inject noise queries for s_i . The simulator chooses the tallies for these queries so they reconstruct to the remaining values in H_{honest} . This guarantees that all correctly handled honest client queries combined with this server's noise have the distribution H_{honest} .

The simulator shuffles the updated query vector and uses it as the output query vector for this server.

3. **If Neither Above Cases are True:** The simulator executes our protocol honestly, including using the same noise queries from the offline stage, to produce the next query vector.

Simulating The Backend: The simulator executes our protocol truthfully, if the backend is not corrupted, or runs the adversary's code if the backend is corrupted, and finds the next response vector.

Simulating Server Online Protocol - Second Pass: The simulator goes through the servers in reverse order, from s_{m-1} to s_1 .

1. **If s_i is corrupted:** The simulator runs the adversary on the current response vector, outputting the next response vector.

2. **If s_i is the first encountered non-corrupted server:**

- **If the backend is corrupted:** The simulator identifies all responses corresponding to honest queries that were mishandled. The responses do not have tags directly embedded in them. However, they should be in the same order as the queries at s_i , which do have these tags. Furthermore, the correct value of the response is known to the simulator, since she can compute it using the T , the value of the corresponding query, and the additive pre-share installed during the offline stage.

The server sends a histogram over the count of these mishandled responses to the ideal function, grouped by their corresponding query value (step 3 in \mathcal{F}).

- **If the backend is not corrupted:** The simulator executes step 3 below.

3. **If s_i is not corrupted:** The simulator identifies all honest queries that were mishandled, using the same mechanism as above. The simulator ignores mishandled queries that were detected in either of the two cases above (the special cases of the first server or backend being corrupted). The simulator only needs to keep count of such mishandled query.

If s_i is the last honest server, she sends this count to the ideal functionality (step 4 in \mathcal{F}).

Simulating Client Online Responses: For every honest client, the simulator checks that her corresponding response, as outputted by s_1 , reconstructs to the expected response value. If the response does not match, then it could have been mishandled by the adversary earlier, and have been already identified by the simulator, such responses are ignored.

The remaining mishandled responses must have been mishandled after the last honest server was simulated. The simulator sends a list of identities of all clients with such responses to the ideal function (step 5 in \mathcal{F}).

Proof. The view of the adversary consists of all outgoing and incoming messages from an to adversary corrupted parties. We show that these messages are indistinguishable in the real protocol from their simulator-generated counterparts.

First, note that all output messages from honest clients in the offline stage are cipher of random values. This is true in both the real and ideal world, and thus these messages are statistically indistinguishable. The same is true for messages corresponding to noise anonymous secrets created by an honest server. The adversary only receives such messages in the offline protocol, and therefore behaves identically in both real and ideal worlds.

Case 1: The backend server s_m is honest.

1. The access patterns are not part of the view, and therefore do not need to be simulated.
2. The corrupted clients are simulated perfectly and has identical outgoing message

distributions in the real and ideal worlds.

3. The honest clients are choosing their queries randomly in the ideal world. However, their messages only include a tag and a tally. The tag is itself selected randomly during the offline stage, and thus has identical distribution. The tally is indistinguishable from random, regardless of the query it is based on, provided that at least one secret share remains unknown, by secrecy of our incremental sharing scheme. In particular, the honest server share is computationally indistinguishable to the adversary from any other possible share value, by CCA security of the onion encryption scheme.
4. The input messages of the first malicious server have indistinguishable distributions in the real and ideal worlds, and therefore the outgoing messages of that malicious server has indistinguishable distributions, since any honest servers prior to this malicious server are simulated according to the protocol perfectly. Inductively, this shows that all malicious servers have indistinguishable distributions during the first pass of the online stage.
5. The backend executes the honest protocol in both worlds. While the backend sees different distributions in either worlds, since honest clients make random queries when simulated, the honest protocol is not dependent on that distribution, and only output responses in the form of secret shares. These secret shares are selected at random during the offline stage by the client, without knowing the response or the query. Therefore, the output of the backend is

indistinguishable in both worlds.

6. Finally, a similar argument shows that the adversary input and output response vectors are all indistinguishable from random in both worlds, since the last secret share of honest queries remains unknown.

Case 2: The backend server s_m is corrupted.

1. The access patterns are part of the view, the simulator must yield a view consistent with them.
2. The outgoing messages of each corrupted client has identical distributions in the real and ideal worlds.
3. The honest client queries are selected randomly. However, they are secret shared. Their secret share component (tally) is indistinguishable from random in both worlds, given that the honest server share is unknown to the adversary. Therefore, their initial tallies are also indistinguishable (but not the access patterns they induce).
4. The input vector to the first server has identical distributions in both worlds, if that server is malicious, then its output vector will also have identical distributions. This argument can be applied to all malicious servers up to the first honest server.
5. The first honest server retains all queries from malicious servers and clients, and handles them as our honest protocol would. However, the server discards all client noise and injects its own queries into it from the provided \mathcal{H} . This

is indistinguishable to the following server from the case where these queries are handled truthfully: (1) the tag component of the query is handled honestly and adversarial perturbations on their enclosing onion ciphers during the offline stage fail due to CCA-security, (2) the tally component of the honest client queries are the result of an incremental reconstruction in our protocol, since the server's share being reconstructed is unknown, the output of this operation is indistinguishable from random even knowing the input. (3) the total count of queries induced by \mathcal{H} is exactly the count of honest client queries that this server discards, plus an amount of noise queries sampled according to the honest noise distribution, this count has the same distribution as the count induced by the honest protocol.

6. The output of the first honest server is indistinguishable, and all the remaining servers are simulated truthfully, therefore their outputs are also indistinguishable., up to the backend.
7. The backend server is corrupted, and can reconstruct the access patterns from the input. However, these access patterns are now indistinguishable between the two worlds, this is because the access patterns of the secret shared queries as outputted by the first honest server in both worlds are indistinguishable: they are both equal to \mathcal{H} + malicious clients and servers queries + mishandled queries. The mishandled queries are guaranteed to reconstruct to random, by our incremental secret sharing non-malleability property, even when their

original queries are different (random in the simulated world).

8. The same argument from Case 1 demonstrates that the view from the second pass of the online stage is indistinguishable in both worlds.

The only thing that remains is to show that the interactions of the simulator and adversary with the ideal function \mathcal{F} are indistinguishable. There are at most 4 such interactions. All of these interactions depend on the simulator's ability to detect when a query or response has been mishandled.

A query may be mishandled by (1) corrupting its tag (2) corrupting its tally by setting it to a value different than the one determined by the associated offline anonymous secrets. Both of these cases can be checked by the simulator, since she has access to the expected uncorrupted anonymous secret values created by every honest client and server. Either of these cases result in the query reconstructing to random, the second case follows from our non-malleability property, the first induces the following honest server to apply an incorrect share when incrementally reconstructing, and thus follows from our non-malleability property as well.

On the other hand, a response can be mishandled by (1) corrupting its tally/value (2) corrupting its relative order within a response vector. The first case arises when an adversary sets the tally value to one different than the sum of its previous value and additive pre-share from its corresponding anonymous secret, as well as when a backend server disregards the underlying database, and assigns a different initial value to a given response. Maliciously perturbing onion ciphers or tallies in the view

of the adversary fall under this case, since this essentially amounts to dropping the corresponding queries, as they are protected by the non-malleability of CCA encryption and our secret sharing scheme. The second case happens when the adversary does not deshuffle responses with the inverse order of the corresponding shuffle. The simulator can check these two cases as well: if a deshuffle was performed correctly, then every response and query at the same index must correspond to one another, and the simulator can compute the expected value of that response from its query value, database T , and additive pre-shares. If the response and query did not match, then either the shuffling or tally computation was corrupted.

We can consider consecutive servers that are adversarially controlled to be a single logical server, since they can share their state and coordinate without restrictions. For example if the first and second server are corrupted, the second server can identify the identities of clients of corresponding to each of its input queries, because the first server can reveal its shuffling order to the second. Similarly with the backend and previous server. This shows that the correct points to check for mishandling is when an honest server is encountered, rather than after every malicious server, since consecutive servers may perform operations that each appear to be mishandling, but consecutively end up handling queries and responses correctly.

Our simulator does the mishandling checks at the level of an honest server. Furthermore, the simulator assumes that any mishandling was done according to the strongest identification method available to the adversary at that point. For exam-

ple, it assumes that the first server always mishandles queries based on their clients identities, even though that server may mishandle queries randomly. In either cases these result in indistinguishable distributions. An adversary that mishandles queries randomly has the same distribution as a simulator that copies that random choice and translates it to identities. No server has the capability to mishandle based on both identity and value, since there must be at least one honest server somewhere between the backend and first server (including either of them).

Finally, intermediate servers (those surrounded by honest servers on both ends) see only query and response vectors that have been shuffled honestly by at least one server, and have a random share applied to their tally by that server as well. So their inputs are indistinguishable from random, and thus they can only mishandle randomly. The first server (and its adjacent servers) see query and response vectors whose tallies are random (because at least one share corresponding to them is unknown), but have a fixed order, since no shuffling has yet occurred, therefore they mishandle queries based on the order (i.e. client identity) as well as randomly. Lastly, the backend (and its adjacent servers) see queries and responses that have been shuffled by at least one honest server, but whose values are revealed, since no shares of these values are unknown. The backend can mishandle queries based on their known value, but not based on their client identity, since mishandling based on index/order is identical to mishandling randomly, because the order is random. □

Algorithm 6 Noise Query Sampling Mechanism $\chi(\epsilon, \delta, \phi)$

Input: The size of the database $|T|$, privacy parameters ϵ, δ , and the number of protected queries ϕ .

Output: A histogram \mathcal{N} over T representing how many noise queries must be issued for each database entry.

1. Clamping threshold $B := \lceil CDF_{Laplace(0, 2\phi/\epsilon)}^{-1}(\frac{\delta}{2}) \rceil$.

For every $i \in |T|$:

2. Sample ϵ -DP Laplace noise: $u_i \leftarrow Laplace(0, \frac{2\phi}{\epsilon})$.
 3. Clamp negative noise: $u'_i := \max[0, B + \min(B, u_i)]$.
 4. $\mathcal{N}[i] = \text{floor}(u'_i)$
-

Differential Privacy Our security theorem contains leakage revealed to the back-end server in the form of a histogram over queries made by honest clients and honest servers. Our privacy guarantees hinge on this leakage being differentially private, which entails adding noise to that histogram from a suitable distribution. Algorithm 6 shows the mechanism each server uses to sample the noise queries \mathcal{N} , and we prove that it indeed achieves (ϵ, δ) -differential privacy. Step (2) is a Laplace substitution $(\epsilon, 0)$ -DP histogram release, which may produce negative values. Step (3) ensures values are non-negative by clamping into $[-B, B]$ and shifting by B , where B is carefully selected in (1) to yield a privacy loss of exactly δ . Table 3.3 shows the expected number of noise queries per server and database element for different ϵ and δ .

Theorem 3.5.2 (Leakage is Differentially Private). $\mathcal{H} = H_{honest} + \chi(\epsilon, \delta, \phi)$ is (ϵ, δ) -

Differentially Private.

Proof. We define (ϕ -)neighboring histograms over access patterns to differ in ϕ or less queries. In other words, no more than ϕ queries from one can be substituted in the other. Therefore, the sensitivity is 2ϕ , corresponding to a change to the first histogram where all ϕ queries are removed from one bin, which thus decreases by ϕ , and added to a different bin, which similarly increases by ϕ . Hence adding noise from $Laplace_{0,2\phi/\epsilon}$ constitutes an $(\epsilon, 0)$ -differentially private histogram release mechanism, this corresponds to value u_i in our mechanism from algorithm 6.

Our mechanism selects B such that $Prob[u_i \leq -B] = Prob[u_i \geq B] = \frac{\delta}{2}$. Note that $u'_i \neq u_i + B$ iff either of these disjoint cases is true, so $Prob[u'_i \neq u_i + B] = \delta$. This implies that using u'_i constitutes an ϵ, δ -differentially private mechanism.

Finally, taking the floor of u'_i is equivalent to taking the floor of $u'_i + c$, where c is the true count of honest queries, since c is guaranteed to be integer. Therefore, floor maintains differential privacy by post-processing. \square

3.6 Scaling and Parallelization

Existing PIR protocols can be trivially scaled over additional resources, by running completely independent parallel instances of them on different machines. This approach is not ideal for our protocol: each instance would need to add an independent set of noise queries, since each reveals an independent histogram of its queries. Instead, our protocol is more suited for parallelizing a single instance over additional

resources, such that only a single histogram is revealed without needing to add ancillary noise queries.

In a non-parallel setting, the notions of a party and a server are identical. For scaling, we allow parties to operate multiple machines. These machines form a single trust domain. This maintains our security guarantees at the level of a party. Particularly, the protocol remains secure if one party (and all its machines) is honest. Machines owned by the same party share all their offline secret state and the noise queries they select.

A machine m_i^j belonging to party j communicates with a single machine m_i^{j-1} and m_i^{j+1} from the preceding and succeeding parties, in order to receive inputs and send outputs respectively. The machine also communicates with all other machines belonging to the same party j for shuffling.

Distributing Noise Generation Our protocol generates noise independently for each entry in the database, we can parallelize the generation by assigning each machine a subset of database entries to generate noise for, e.g. m_i^j is responsible for generating all noise queries corresponding to keys $\{k \mid k \% j = 0\}$. This distribution is limited by the size of the database. If parallelizing the noise generation beyond this limit is required, an alternate additive noise distribution (e.g. Poisson [236]) can be used instead, which allows several machines to sample noise for the same database entry from a proportionally smaller distribution.

Distributed Shuffling Machines belonging to the same party must have identical probability of outputting any input query after shuffling, regardless of which server it was initially sent to. An ideal shuffle guarantees that the number of queries remains uniformly distributed among machines after shuffling. We choose one that requires no online coordination to ensure it maintains perfect scaling. Machines belonging to the same party agree on a single secret seed ahead of time. They use this shared

seed locally to uniformly sample the same global permutation P using Knuth shuffle. Given a total batch of size l , each machine m_i^j need only retain $P[\frac{i \times l}{m} : \frac{(i+1)l}{m}]$, which determines the new indices of each of its input queries. The target machine that each query q should be sent to can be computed by $P[q] \% \frac{l}{m}$. This algorithm performs optimal communication $\frac{l}{m}$ per machine but requires each machine to perform CPU work linear in the overall number of queries to sample the overall permutation. This work is independent of the actual queries, and can be done ahead of time (e.g. while queries are being batched or processed by previous parties).

Distributing Offline Anonymous Secrets We require all machines belonging to the same party to share all secrets they installed during the offline stage, so that any of them can quickly retrieve the needed ones during the online stage. Maintaining a copy of all secrets in the main memory of each machine may be suitable for smaller applications. At larger scales, it may be more appropriate to use shared key-value storage or in-memory distributed file system [33, 167, 188, 255].

3.7 Evaluation

Experiment Setup Our various experiments measure the server completion time for a batch of queries. For the online stage, this is the total wall time taken from the moment the first server receives a complete batch ready for processing, until that batch is completely processed by the entire protocol, and its outputs are ready to be sent to clients. For the offline stage, the measurements start when the complete batch is received by the first server, and ends when all servers finished processing and installing the secrets. Measurements include the time spent in CPU performing various computations from the protocol, as well as time spent waiting for network IO as messages get exchanged between servers. Our measurements do not include client processing or round-trip time.

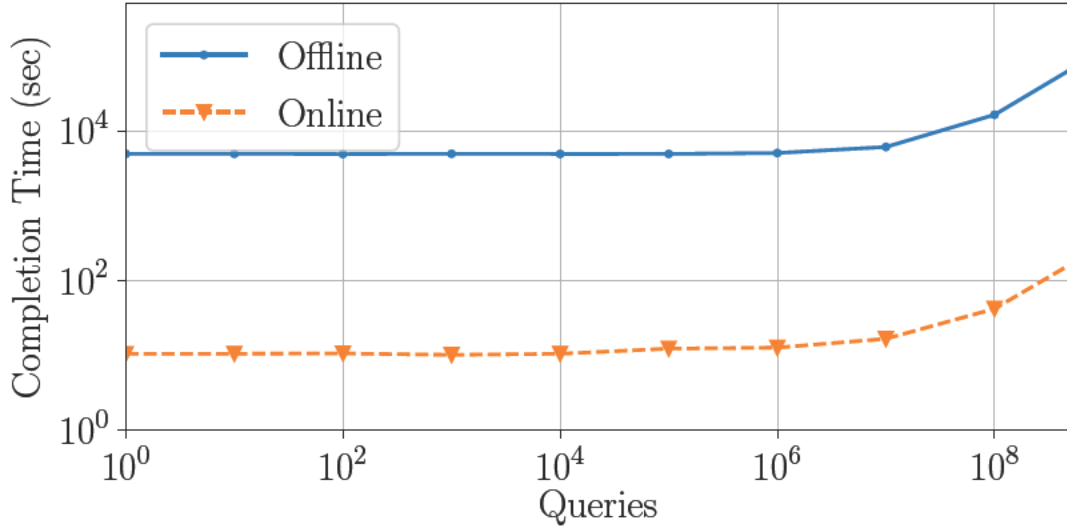


Figure 3-7: Completion time for varying number of queries against a 100K database (logscale)

All experiments in the paper use $\epsilon = 0.1$ and $\delta = 10^{-6}$. Keys and values in our database are each 4 bytes, with signatures that are 48 bytes long (e.g. BLS [58]). We ran our experiments on AWS r4.xlarge instances that cost around \$0.25 per hour, using only one thread. A primary factor in selecting these instances is RAM, since we need sufficient memory to store large query batches. We implemented our protocol using a C++ prototype with about 6.1K lines of code. Our prototype relies on lib-sodium’s `crypto_box_seal` [99] for encryption. Our code is available on GitHub [104].

Scaling Figures 3-7 and 3-8 show how our protocol scales with the number of queries and database size, respectively. Our runtime is dominated by noise queries when the number of queries is smaller than the size of the database, and begins to increase with the number of queries as they exceed it. For a large enough number of queries, our runtime scales linearly as the overhead of noise queries is amortized away over the real queries. Our noise overhead scales linearly with the size of the database. The cost of processing any input query *in isolation (without noise)* is constant and does not depend on the database size, which only affects the number of noise queries

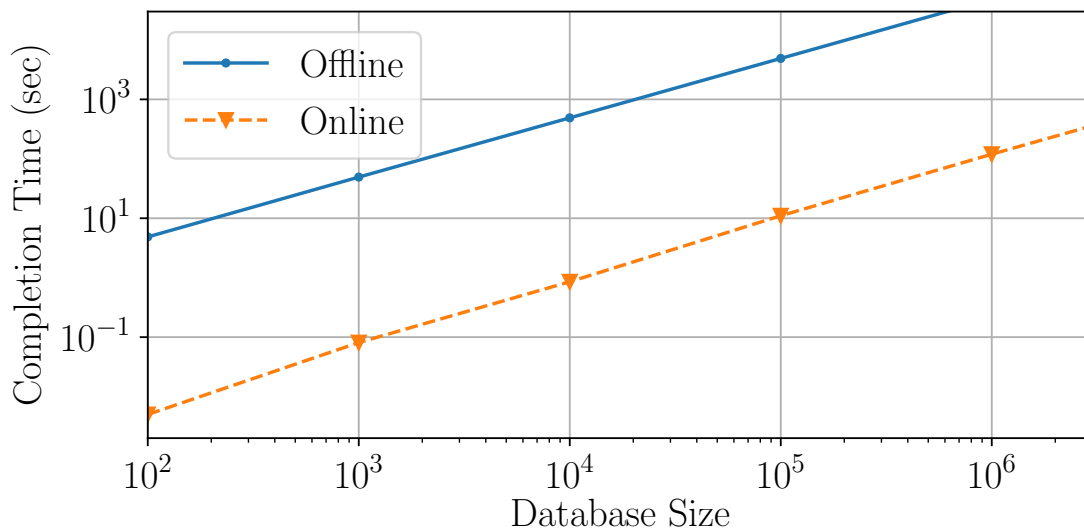


Figure 3.8: Completion time for a batch consisting only of noise queries against varying database sizes (logscale)

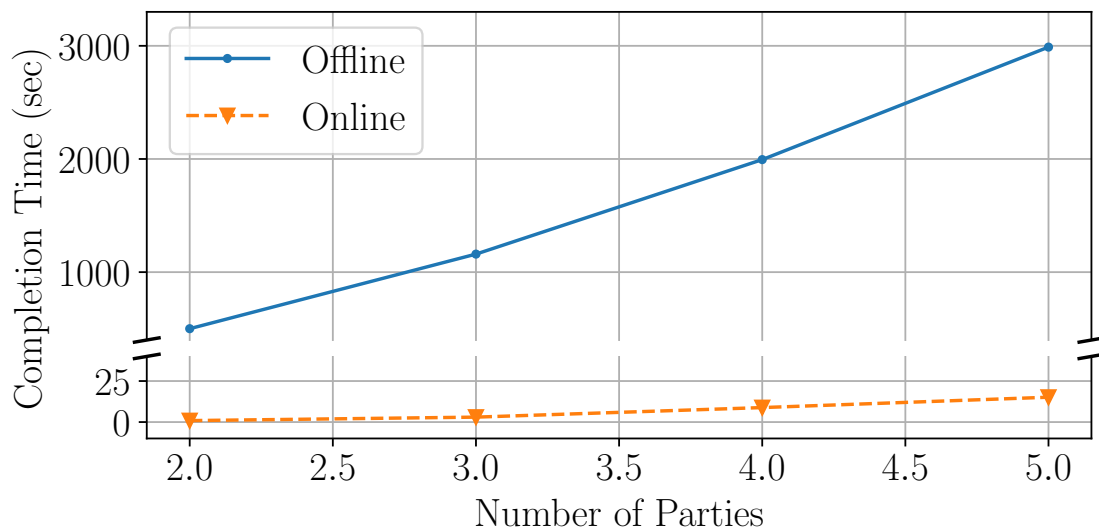


Figure 3.9: Completion time for varying number of parties with 100K queries against a 10K database

added by our protocol. The offline stage is about 500x more expensive than our online stage. This is expected since the offline stage performs a public key operation for each corresponding modular online arithmetic operation.

Machines / Party	Server time (seconds)	
	Offline	Online
1	5010	11
2	2560	8.2
4	1296	4.0
8	664	2.2

Table 3.2: Horizontal scaling with 1M queries and a 100K DB

$\delta \backslash \epsilon$	1	0.1	0.01	0.0001
10^{-5}	23	230	2302	23025
10^{-6}	27	276	2763	27631
10^{-7}	32	322	3223	32236

Table 3.3: Expected number of noise queries B per database element as a function of different ϵ (columns) and δ (rows)

Figure 3.9 shows how our protocol scales with the number of parties. Our protocol is most efficient when only two parties are involved. When the number of parties increases, a query has to pass through more servers as it crosses the chain. This is more pronounced in the offline stage, as it additionally increases the size and layers of each onion cipher, causing the offline stage to scale super-linearly in the number of parties. In addition, each server naively adds the full amount of noise queries required to independently tolerate up to $m - 1$ corrupted parties. Adding less noise by relying on additional assumptions (e.g., honest majority) is an open problem, which can help improve our scaling with the number of parties, and can have important consequences to mixnets, the DP shuffling model, and DP mechanisms in general. Techniques such as noise verification [172] may be useful to ensure that (partial) noise generated by an honest server is not tampered with by future malicious servers.

Table 3.2 demonstrates how our protocol scales horizontally. Parallelizing the online stage primarily parallelizes communication. However, parallel shuffling introduces an additional round of communication per party. As a result, our online stage speed up when using 2 machines is not 2x. We exhibit linear speedups as the number

of machines exceeds 2.

Finally, the expected number of noise queries added per database element is a function of ϵ and δ . Table 3.3 lists this expected number for various combinations of ϵ and δ . The expectation increases linearly as ϵ decreases but scales better with δ . This means that the amount of noise overhead (and thus the number of queries required for that overhead to amortize effectively) grows linearly with $\frac{1}{\epsilon}$. Our protocol trades security for performance. It can efficiently amortize the cost of independent queries due to its relaxed DP security guarantees. As ϵ becomes smaller, this relaxation becomes less meaningful, as the DP security guarantees approach those of computational security. While linear scaling with $\frac{1}{\epsilon}$ appears to be intrinsic to our protocol, we believe it may be possible to reduce the scaling constant, by using different basis distributions that are inherently non-negative or discrete (e.g. Poisson [236] or Geometric [186]), or by adapting recent work on privacy amplification [81, 107] that achieves the same level of privacy using less noise with oblivious shuffling.

Latency Latency in Checklist and similar systems includes the computation cost of a single query *in isolation* (which is low), and any *queuing delays* experienced by the query after its arrival if the computational resources are busy handling previous queries. This delay depends on the rate at which queries come in, and can be significantly larger than the batching overheads in applications with a large query load. In contrast, our protocol is primarily throughput oriented and its latency is a secondary concern determined by two components: the idle waiting time required to collect the batch of queries from different clients, which we call the *batching window*, and the active processing time of that batch after collection. The first component depends on the configuration. The later component is precisely the total computation time measured in the various experiments in earlier parts of the paper. Lowering the batching window beyond a certain point can have a negative impact on latency (and

even throughput), since it can result in smaller batches dominated by noise where amortization is not effective. Furthermore, it can introduce queuing delays at the level of batches, where a previous ongoing batch still occupies system resources after the next batch has been collected.

We summarize three important observations: (1) Queuing delays in existing systems are significant and can cause them to exhibit latency worse than DP-PIR with a large number of queries. (2) Both DP-PIR and existing systems can be scaled horizontally to exhibit lower latency. Traditional PIR protocols can achieve sub-second latencies if given enough resources, but this can be prohibitively expensive when the query rate is high. (3) For our target large query loads, DP-PIR can be configured to exhibit decent latency with a much lower budget than existing systems.

The Offline Stage PIR protocols with an offline stage typically do so to improve their online latency, which is less critical in our target applications. It is possible to combine both DP-PIR stages into a single stage that performs onion-encryption of the query directly, without the need to install anonymous secrets. This combined protocol would exhibit similar trends to our current design, but will be around two orders of magnitude slower than our online protocol on its own. A fair comparison here must also account for the offline cost of existing protocols, which can be significantly larger than our offline cost. For example, Checklist relies on an expensive per-client offline stage linear in the size of the database, which we observe takes up to 7 seconds per client in our experiments. In DP-PIR, the offline cost for a single query amortizes to a few milliseconds. One key difference is that a client can reuse the hint produced by Checklist’s offline stage to make many following queries, rather than a fixed number of queries in DP-PIR. However, the hint becomes invalid whenever the database is updated. Checklist provides an updatable offline construction, where a single update to the database can be carried over to a previous offline computation

in cost logarithmic in the database size.

We believe that the offline-online design provides better deployment cost and performance, and allows DP-PIR to meet the availability and liveness requirements of many applications, including our App store example. Concretely, the offline-online design allows greater control over the batching window, which governs the effectiveness of amortization, client latency, and the duration needed for updates to the database to become visible to clients at the next batch. For example, it may be desirable to allow clients to query the App store multiple times a day, e.g. every hour, in order to discover important app updates earlier. A natural way to achieve this is to use a batching window of one hour or less. However, this is only effective if this window includes sufficient queries for amortization, and has sufficient time to complete processing before the next batch. The offline setup lowers both requirements, making smaller windows practical (or alternatively, cutting the online cost of the same window by 500x).

The offline stages for multiple online stages can be pooled together and executed ahead of time. Clients can choose to issue less queries than they signed up for in the pooled offline stage without privacy loss. Service providers can use this to execute the combined offline stages during off-peak hours when resources are cheaper (e.g. overnight). Furthermore, providers can use different setups for each stage to optimize the effectiveness of their budget. The offline stage is CPU-intensive due to its public key operations, while the online stage is entirely network bound.

3.8 Related Work

Section 3.2 discusses existing work on Private Information Retrieval. Here, we discuss related work from other areas.

Mixnets Traditional mixnets [78] consist of various parties that *sequentially* process

a batch of onion-ciphers, and output a uniformly random permutation of their corresponding plaintexts. Various Mixnet systems [50, 119] add *cover traffic* to obfuscate various traffic patterns. However, ad-hoc cover traffic is shown to leak information over time [185].

Recent work mitigates this by relying on secure multiparty computation [26] or differential privacy. Vuvuzela [243] adds noise traffic from a suitable distribution to achieve formal differential privacy guarantees over leaked traffic patterns, and Stadium [236] improves on its performance by allowing parallel noise generation and permutation. Similar techniques have been used in private messaging systems [172], and in differential privacy models that utilize shuffling for privacy amplification [107] or for introducing a *shuffled* model that lies in between the central and the local models [81].

Differential Privacy and Access Patterns Using differential privacy to efficiently hide access patterns of various protocols has seen increasing interest in the literature. ϵ -PIR relaxes the security guarantees of PIR to be differentially private [233] in the semi-honest setting. Their two AS schemes are closest to our protocol: they require clients (rather than servers) to generate noise queries along with their real queries, and send all of them through an anonymous network for mixing. When the number of clients is large enough, this can amortize the number of queries any of them have to generate to a constant. However, this approach generates far more total load on the system. For example, in our app store example with 2 servers, a 2.5M database, and 3B clients, each client needs to generate 282 noise queries to hide a *single* query with $\epsilon = 0.1$, which results in close to $850B$ queries to the system in total, compared to the $< 4B$ total load on our system (but with $\delta = 10^{-6} \neq 0$). These constructions do not provide integrity guarantees, and will require further noise queries to protect against potential malicious or unavailable clients.

Others relax the security of Oblivious RAM (ORAM), a primitive where a single client obliviously reads and writes to a private remote database [124, 125], to be differentially private. Extensions of ORAM address multi-client settings [182]. Differentially oblivious RAM [76, 247] guarantees that neighboring access patterns (those that differ in the location of a single access, i.e. event-DP) occur with similar probability. DP access patterns have been studied for searchable encryption [79] and generic secure computation [186].

Secret Sharing Shamir Secret Sharing [221] allows a user to split her data among n parties such that any t of them can reconstruct the secret. Secret sharing schemes with additional properties have been studied for use in various applications. Some schemes, such as additive secret sharing, allow the secret to be reconstructed *incrementally* by combining a subset of shares of size k into a single share that can recover the original secret when combined with the remaining $n - k$ shares. *Non-malleable* secret sharing schemes [37, 127] additionally protect against an adversary that can tamper with shares, and guarantees that tampered shares either reconstruct to the original message or to some random value. Aggarwal et al. [22] show generic transformations to build non-malleable schemes from secret sharing schemes over the same access structure.

3.9 Conclusion

This paper introduces a novel PIR protocol targeted exclusively at applications with high query rates relative to the database. This focus is intentional and necessary: DP-PIR handles large batches so well specifically because it handles small ones poorly. Our construction makes PIR usable in scenarios that were previously impractical or unexplored. DP-PIR is primarily geared towards amortizing total server work (i.e. throughput), but not for sub-second client latency, and only provides relaxed

differential privacy guarantees.

The performance of DP-PIR is closely tied to its configurations, which determine the number of noise queries generated by our system, and thus the number of queries required to amortize their overheads effectively. Our experiments meet or extend beyond standard configurations suggested by existing work. Checklist [162] supports exactly two parties, and PIR schemes are rarely instantiated with more than three. For small databases (e.g. $n < 100K$), the naive solution of sending the entire DB to the client may be desirable. Vuvuzela [243] recommends $\epsilon \in [0.1, \ln(3)]$ and sets $\delta = 10^{-4}$, and other work [186, 233] also mostly focuses on $\epsilon \geq 0.1$.

The ratio of queries to database size $\frac{q}{n}$ is the primary performance criteria that governs how effective DP-PIR is compared to existing protocols. Within the space of *typical configurations* outlined above, our experiments demonstrate that applications with $\frac{q}{n} < \frac{1}{10}$ are unsuited for DP-PIR, while applications with $\frac{q}{n} > 10$ are almost always guaranteed to exhibit speedups of several folds when using DP-PIR. Applications with ratios in $[\frac{1}{10}, 10]$ may or may not be suited to DP-PIR, depending on their exact configurations. For example, we can achieve better performance than existing work for a ratio of 0.8 when the database size is $2.5M$, but not when it is of size $1M$ (section 3.2). Thus, such applications require individual analysis to determine the best way to realize them.

Our protocol shifts expensive public key operations to an offline stage. This allows for more flexibility over the batching window to meet application requirements, and a more efficient allocation of computational resources. However, applications where these factors are not a concern may elect to combine the two stages into a single one, that still exhibits similar trends to our online stage, but is about two orders of magnitude more expensive. Finally, these ratios, and the number of noise queries, also depend on the level (and duration) of protection offered to users (e.g. event-DP

vs user-time-DP) as expressed by ϕ . DP-PIR intentionally relaxes its guarantees for increased performance. This relaxation becomes less meaningful as ϵ and ϕ approach perfect security.

Chapter 4

Cryptographic Tools for a Wider Audience

In this chapter I go over some of my projects that helped further the adoption of secure computation in practice, through extensive collaborations with the various communities the privacy tools were designed for. All of these projects went through an iterative design process with feedback from the target community and had similar flavors of challenges:

1. They required some level of “de-mystification” of the secure computation to non-experts so that we could instill trust in these tools.
2. They had to carefully consider the different resources and time restrictions available to different parties in the computation.
3. They handled human errors in incorrectly formatted data to the computation.
4. The end software delivered to the community had to be usable and accessible by non-experts.

Its important to clarify what I mean in this chapter by “usable and accessible”. There is a long line of beautiful and essential research in usable security [16–18, 38, 74, 96, 100, 113, 123, 126, 133, 136, 159, 193, 201, 209, 232, 251, 252, 257] that aims to systematize properties and techniques for making secure and privacy preserving tools more usable by their intended audience. This kind of work is integral to the goal of ensuring that privacy reaches a wider audience and making sure that

people understand how to use these tools (e.g. password specification, security warnings, etc.). However, the challenges and solutions I present in this chapter do not directly fall into this line of research. My projects were carefully tailored for their intended community and require more work before they can be deployed at scale while maintaining their usability and accessibility features. It's worth noting that in the case of my work with Boston's Women's Workforce Council and the Greater Boston Chamber of Commerce, my co-authors went on to do precisely some of this work in systematizing ways of making MPC more usable [211].

4.1 Private Evidence Based Policy Making at the Department of Education

In this section, I describe a pilot [2,6,8,35] I have worked on for the U.S. Department of Education to demonstrate that their common statistical studies can be performed privately using MPC without a trusted third party contractor and at very little additional resource costs. I was in charge of all technical aspects of this project with occasional advice/guidance from Dave Archer and Brent Cramer at Galois. We also collaborated with Stephanie Straus and Amy O'Hara from the Georgetown's Massive Data Institute on the policy and data science aspects of the project.

4.1.1 Background

In 2017, the US Commission on Evidence-Based Policymaking [11, 12] unanimously recommended that inter-agency sharing of administrative data should be accompanied by enhanced privacy protections. Later in 2019, Congress heeded the recommendations of the Commission and revised the earlier Confidential Information Protection and Statistical Efficiency Act of 2002 by enacting the Evidence Act [4]. This bill pushes federal agencies to modernize their data sharing, collection and analysis processes in order to inform better policy making decisions. Most notably, the Evidence

Act requires agencies to protect the confidentiality of participants in statistical studies while promoting data sharing among agencies. Unfortunately, current approaches for doing so often involve outsourcing any computation that is performed on said data to third parties that are contractually obligated to safely and securely handle the data.

4.1.2 Our Work and Design Goals

Our pilot demonstrates to the Department of Education how MPC can efficiently and securely perform any statistics needed for evidence base policy making in-between agencies with no recourse to anyone outside the department itself and without any privacy risks. The Department of Education instructed us to securely reproduce a portion of the annual 2015–16 National Post-secondary Student Aid Study (NSPAS) [117], a commonplace statistic performed by its National Center for Education Statistics (NCES) division. This study determines how post-secondary students across the U.S. finance their education in order, for instance, to inform how to allocate federal financial aid funds in subsequent years.

Currently, NCES compiles this study by hiring a third party contractor, who we will call NPSAS for simplicity, to perform the necessary computation since federal law prohibits it from holding individual student records. NPSAS starts by selecting students that will contribute to the study based on their social security number (SSN) and assigning them weights that determine how representative they are of the overall population. NPSAS then queries the National Students Loan Data System (NSLDS) with the selected SSNs to obtain relevant federal student aid information for each. NPSAS then categorizes the data and computes a weighted average per category and sends the resulting report to NCES. In the process, NSLDS learns which students were selected by NPSAS, while NPSAS learns a significant amount of sensitive information pertaining to each student like their enrollment status, their family’s financial information, their loan forbearance data, their federal financial aid across

several academic years, their contact information, etc.

The primary goal of the pilot was to demonstrate to the Department of Education that MPC could eliminate this leakage and maintain the guarantees outlined by the Evidence Act, or in other words that MPC (1) could be practical and usable, (2) could produce correct and reproduceable results, (3) could be efficient and (4) eliminates the need for outsourcing the computation to a trusted third party.

4.1.3 Practicality and Usability

To showcase the practicality of MPC, our protocol and software had to rely exclusively on the technical infrastructure available to NPSAS and NSLDS. To that effect, we tested our work in an environment that emulated the environments of NPSAS and NSLDS prior to the time of the demo. We specifically tested our work on an Amazon EC2 r5.4xlarge instance with 128GB of RAM running Microsoft Windows Server 2016 Datacenter edition, and an Amazon EC2 r5.2xlarge instance with 64GB of RAM running Microsoft Windows Server 2012 edition, in both cases using 4 cores of the underlying Intel Xeon Platinum 8259CL, 2.50GHz CPUs.

The goal of usability informed several aspects of our work. First, the pilot was demonstrated and tested on real student records by a policy fellow with no prior cryptographic knowledge. This meant that the software that I developed had to be easy to use by someone with little technical expertise and easy for me to remotely debug with no access to the underlying data or the machine on which the demonstration was run. All that was asked of the policy fellow was to: (1) fill out a text file where they would specify the directory of the student records and an upper bound on the number of entries in the data, and (2) run an executable. The cryptographic intricacies of the code were hence completely hidden from them. Second, a large portion of this project's time was dedicated to explaining to the policy fellow how MPC and specifically Private Set Intersection (PSI) works in order for them to later

convey these core concepts to the Commissioner of the NCES during the period of demonstration. This was an essential part of explaining to policy makers and lawyers why MPC in this context would not constitute a disclosure under FERPA in order to later certify it as non-disclosing.

4.1.4 Correctness and Reproducibility

While it may be intuitive to cryptographers that an MPC computation designed for a specific functionality can be provably “correct” and reproducible, we still had to exhibit these properties to the Department of Education and test the correctness of our program. The policy fellow compared the outputs of our MPC protocols with the “in the clear” computation of the Department of Education. At first we observed a disparity in the results because of an incorrect specification from NCES of how the NPSAS report is computed. It was non-trivial to determine the source of this disparity since we initially had to check that gates in the MPC circuit were behaving correctly before realizing that the error was not from our implementation. The true error came from a misunderstanding of how exactly STATA, a proprietary software, computes averages. After rectifying our functionality, we found minimal differences that amounted to rounding errors. Additionally, only the policy fellow had the required clearance to handle the student records. As such, we synthesized data-sets from public parameters from the 2016 study report [117] (e.g. standard deviation, mean, etc.) to illustrate the reproducibility of our protocol and test our software prior to the time of the demonstration.

4.1.5 Protocol Choice and Performance

For our MPC protocol, we implemented Pinkas et al. [207] as the basis of our pilot because it offered the best combination of computational and communication efficiency and computational expressiveness. Since NCES did not disclose how they computed

the results of the National Post-secondary Student Aid Study until near the end of this project, we opted for a generic PSI framework that could encompass any possible computation. While we may have picked a more efficient PSI construction, the timeline of this work did not allow us to do so, especially since the window of access to the real student data was very brief. A protocol that allowed for fast development and easy modifications to the underlying computation was key to the success of this project. Pinkas et al. is roughly structured as a PSI protocol followed by a circuit over the elements of the intersection and their associated data. Modifications to the computation in our case translated to changes in the circuit.

In terms of performance, we had to find the right balance between communication and computational efficiency versus ease of making changes rapidly as we learned more about the Department of Education’s needs. Recent years have seen large scale deployments of PSI such as Ion et al. [144], which optimizes for monetary costs and communication effectiveness but provides only a limited set of statistics that can be computed over the associated data; and Buddhavarapu et al. [67], which optimizes for setups where intersected identifiers are not readily at hand and may be re-used. Neither of those protocols could suit our needs because they did not provide the right trade-off between efficiency and expressiveness. While industrial deployments of PSI typically prioritize minimizing bandwidth, we had to demonstrate to NCES that the MPC computation would not take too long as per their specifications. With our optimizations, a test with 1 million synthetically-generated records per party ran in 30 minutes and used 100 GB of communication, and the real demonstration (on an unknown size) ran in 4.8 hours.

At a high level, our circuit amounted to computing the weighted average of elements in the intersection of SSNs. Our optimizations were hence three-fold.

First, we used techniques from Ball et al. [39, 40] to represent the circuit arith-

metically and lower the communication costs during the exchange stage of the garbled circuit. Ball et al. has the added advantage of rendering addition gates for free. Hence, the computational bottlenecks in these circuits are due to multiplications and comparisons, and these gates can be made more efficient, as we did, using mixed-moduli representations. The weighted average circuit only contains a single final division gate which did not hamper the efficiency of the protocol.

Second, because computational efficiency was essential, we parallelized the computation of the circuit and “soldered” each thread’s output in a joining circuit that computed the final result. This overall increased the communication of the protocol but vastly improved the runtime of our code. Finally, we implemented our software in Rust because it allowed for better performance and memory safety. However, we first had to get clearance for Rust as a “software” before being able to run our code within the Department of Education. This meant that we had to additionally explain to the NCES representatives why we were using Rust, why it was a safe language and how it was not only a standard but also best practice programming language for cryptographic use cases. Our implementation uses the Rust MPC framework *swanky* [122], and it is available open source on GitHub [5].

4.2 Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities

In this section, I describe a web-based MPC system I worked on for the Boston Women’s Workforce Council and the Greater Boston Chamber of Commerce that enabled them to perform their studies privately. I was part of a larger team that developed, designed and deployed these tools. I was also one of the two PhD students who created and designed JIFF. This section is based on joint work [169, 171] with Kinan Dak Albab, Andrei Lapets, Frederick Jansen, Peter Flockhart, Lucy Qin, Ira

Globus-Harris, Wyatt Howe, Mayank Varia and Azer Bestavros.

4.2.1 Background

In 2014, the Boston Women’s Workforce Council (BWWC) was created by the City of Boston to advance the interest of women in the workplace. One of its aims is to close the wage gap between genders, races and ethnicities. Their 100% Talent Compact initiative now includes over 250 businesses that collectively measure the wage gap and make the necessary changes to close it. From the start of this project, a fundamental challenge presented itself: no entity was willing to handle the wage data as a trusted third party because of the legal liability involved.

4.2.2 Web-MPC: Our Work and Design Goals

During my time at the Software & Application Innovation Lab (SAIL) at Boston University, we helped the BWWC address this concern by developing and deploying an MPC system called Web-MPC that computes all the BWWC’s statistics without a trusted third party. Web-MPC takes into account the asymmetric roles and resources available to parties and the various needs of the BWWC. The BWWC has since been able to run the analysis securely every 1-2 years since 2016. Subsequently, the Greater Boston Chamber of Commerce (GBCC) requested a similar deployment from us and launched the Pacesetters Initiative in 2018 where they could securely measure spending with Minority Business Enterprises (MBEs) in the greater Boston area.

The success of both of these projects stems from requirements that are very similar to the use case of the Department of Education, as we will soon see. The synthesis of these properties in both the BWWC and Department of Education cases, all originated from extensively interacting with the people for whom the cryptographic tools were being designed and built. Both of these cases would not have succeeded without involving policymakers and stakeholders into the process of designing and deploying

our MPC protocols. In particular, it was essential to explain our protocols and their guarantees to these people in the most accessible way possible.

4.2.3 Practicality and Usability

One of the first barriers that we encountered in the BWWC project surrounded the question of practicality. Different parties have access to different resources, and no other party on the BWWC's end had the computational resources needed to enable or catalyze any possible MPC deployment. By contrast, we at SAIL were both willing and able to fill this computational gap. This asymmetry of resources was the motivating factor in the architectural design of our protocol. Using MPC, we could outsource the computation to SAIL without ever requiring anyone at SAIL to hold the data at rest. As a result, we designed the MPC protocol so that the client-side software would share the companies input data between SAIL and the BWWC's data analyst which then handle the computation.

Moreover, the BWWC's analysis could take several weeks to conclude because a larger data collection window incentivized more businesses to participate in the study. In other words, our protocol had to take into account that the BWWC's data analyst was only available to partake in the MPC protocol at the very end of the data collection time-frame, and the businesses asynchronously shared their data. To that effect, we required businesses to submit both their shares to us in such a way that the BWWC's share was encrypted under the BWWC's public key. When the data collection period was complete, we would hand out the encrypted shares to the BWWC data analyst and begin the MPC protocol.

Similarly to the Department to Education, we could not assume any specialized software or hardware capabilities for the parties in our protocol. All business who were participating in the 100% Talent Compact initiative had various software restrictions on company owned devices. The only software that they all had in common were web

browsers and Excel spreadsheets. As a result, both the BWWC and GBCC use cases suggested the need for a general purpose web based framework for MPC.

As such, we developed the Javascript-based JIFF [93] to support MPC applications that (1) run on web and mobile platforms in which parties join and leave the computation dynamically, (2) require that computations occur asynchronously, and (3) require mechanisms for recovery from network and crash failures. JIFF is highly customizable to accommodate the idiosyncrasies of these kinds of deployment scenarios. It can be easily integrated within server-client(s) web and mobile applications, and server(s)-to-server(s) systems, in which the essential non-MPC features have been built by developers who do not necessarily possess extensive cryptographic know-how. In addition to the BWWC and GBCC studies, JIFF has been used in two other notable applications by other groups: (1) a secure accountability of electronic surveillance system for the US federal courts [118], and (2) a decentralized and encrypted gun registry [155]. By integrating JIFF into Web-MPC, we were able to easily compute all statistics desired by either the BWWC or the GBCC, including standard deviations and finer resolution statistics grouped by cohort or sector.

4.2.4 Correctness and Error Handling

One thing to highlight here is that the system we designed for both the BWWC and GBCC was constructed to handle common failures, and with end-user usability front and center. Our experience demonstrates that usability is a *critical* requirement that governs whether a particular application of MPC is successful. Usability here extends from the cryptographic building blocks (e.g. sharing scheme) and how easily they can be explained to non-technical stakeholders to inspire trust in the protocol, to the design of the protocol itself (e.g. the asynchronosity mentioned earlier), and up to the user interface for data entry and for retrieval of aggregate statistics.

Although many of these usability concerns are similar to non-MPC traditional

software applications, they take on new dimensions given the cryptographic nature of MPC. One crucial complication we highlight is handling human error in input data. Traditional MPC protocols do not reason about the syntactic or semantic correctness of inputs to the protocol, even though such errors can have a catastrophic impact on the fidelity of the computation and its outputs. Unlike insecure computations, where inputs are available for manual inspection (and if need be, correction), it is hard to detect or correct such errors when using MPC, as all inputs are secret and unavailable for manual inspection. In fact, during one of the earlier deployments of GBCC, some users entered incorrect data because of bad unit conversions. The results of the computation was orders of magnitude larger than it was expected to be. It was infeasible to ask participants to resubmit their data and we could not pinpoint the source of the error because of the input anonymity guarantees of MPC. Instead, we had to develop a custom MPC protocol using JIFF that processed the stored additive shares submitted by input parties, transformed them into Shamir secret shares, compared each share under MPC against some heuristics we designed given our suspicions about what the errors could be, and then attempted to correct these errors under MPC if the heuristics were a match, e.g. by performing the unit conversion ourselves. This process was extremely tedious: we had to develop this correction protocol while “flying blind”, because we could not see the data. It also was expensive to run, taking several hours to complete, and required the involvement of the GBCC as they controlled one share of each value.

Given this experience, we improved the UI of Web-MPC for future deployments to perform several syntactic checks (e.g. ensuring values are within specified ranges), and consistency checks across the input spreadsheets. Our UI highlights to users the violated checks in red or yellow depending on the severity of the violation, and displays various summaries to users that they have to check and agree to prior to submission.

This was a continuous effort that spanned multiple deployments and that some of my collaborators later formalized and extended in a paper on the usability of secure computation [211].

4.3 Carousels: Tool chain support for emerging cryptography

In this final section, we direct our attention towards ongoing work I have on automated resource estimation of MPC protocols, in order to further the adoption of MPC tools by improving its toolchains to better support developers. This section is based on joint work with Kinan Dak Albab, Peter Flockhart and Wyatt Howe.

4.3.1 Background

Developers currently face several unique challenges when developing, maintaining, and reasoning about MPC software. The MPC paradigm differs greatly from its insecure counterpart, with seemingly well known operators and constructs taking on new security and performance properties that sometimes depend on its cryptographic realization. Furthermore, there is a large space of MPC protocols and implementations each associated with its own configurations, security guarantees, and performance implications. Meanwhile, the MPC ecosystem lacks the software abstractions and toolkits that aids developers in navigating the tangled multi-dimensional space of protocols, configurations, security, and performance, especially for non-experts. These challenges and the need for better abstractions and tooling support has been highlighted in the literature [36, 138, 169, 204]. Here, we will focus specifically on assisting developers in reasoning about the performance of their MPC implementations and in applying optimizations to improve it.

Reasoning about the performance of secure programs is challenging for developers. Different secure primitives have radically different costs, even when their insecure

counterparts are similar (e.g. $+$ and \times). Secure frameworks [30, 54, 93, 98, 161, 187, 190, 222, 245, 249] provide transparent abstractions that purposely hide cryptographic details, and thus do not reflect underlying costs. Finally, the performance of secure programs is governed by domain-specific resources, including communication round complexity in secret sharing, or the noise growth in homomorphic encryption. These resources depend on the program control flow, specifically its critical path or multiplicative depth. As a result, the optimal secure algorithm and implementation for a certain problem and a specific region of the configurations space may differ from an insecure one, especially given the oblivious control flow of secure programs.

The high dimensionality of the secure protocols and configurations space has well-known negative consequences on the ease of developing and optimizing secure solutions. Existing work [36, 204] shows that secure protocols can vary along several dimensions that influence their applicability and usability to various application scenarios, and that relate to setup (e.g. the number of parties, the use of pre-processing), security (e.g. the tolerated collusion threshold), and performance (e.g. online communication rounds per primitive). In practice, the number of dimensions becomes even larger when framework implementation and architecture dimensions are accounted for [138].

As a result, developers must implement secure programs while co-reasoning about correctness, security, performance and all the above dimensionalities. This process is particularly painful because of three unique observations. First, the performance and resources used by a secure program are governed by an entangled web of unfamiliar parameters. Second, the programming abstractions provided by secure frameworks are purposely opaque, and thus do not reflect the underlying costs nor the actual execution of the program. Finally, the wall-time performance of secure programs as experienced by end-users is heavily influenced by non-traditional resources such as

network rounds and noise levels in ciphertexts, which depend on how the program is executed rather than how it appears as written.

4.3.2 Motivating Example: Battleships

We demonstrate these challenges by analyzing the resource usage of three implementations of the game Battleships with square boards of length N , shown in Figure 4.1. In this thought experiment, we will play the role of a developer attempting to implement a secure Battleships game via the BGW [49] secret sharing-based protocol with two distrusting players and a server. For simplicity, we do not show the secret sharing or reconstruction stages, but instead focus on the comparison step between guesses and real locations. The guess or locations of a player’s ships cannot be revealed to the other player or untrusted server. Instead, comparing the guesses and locations must be done securely such that only the matching locations are revealed.

A reasonable starting implementation represents each secret guess and ship location as a pair of secret x and y coordinates on the grid, as shown in `battleships1`. For a *single* guess, `battleships1` executes $|ships|$ secure ANDs and XORs, and $2 \times |ships|$ secure equality checks. The equality checks are all independent and can be performed concurrently, and secure XORs are cheap and can be computed without communication. A single secure AND or equality check can be carried out in 1 and $b - 1$ network rounds respectively, where b is the number of bits in the underlying finite field, making the entire program executable in exactly b rounds. However, note that the field needs to be large enough to represent the input values. Thus b must be at least $\lceil \log(N) \rceil$. On the other hand, the total bandwidth exchanged between the players is in $O(|ships| \times b^2)$ per guess, since each comparison sends $b - 1$ messages each of size b .

A logical subsequent refinement is to reduce the number of comparisons executed overall, such as in `battleships2`. Instead of checking equality of coordinates, we

```

fn battleships1(guesses: Vec<Pair<Secret<int>>>,
                ships: Vec<Pair<Secret<int>>>) -> Vec<Secret<boolean>> {
    let mut result = Vec::new();
    for gx, gy in guesses {
        let matched = 0;
        for lx, ly in ships
            matched ^= (gx == lx && gy == ly);
        result.push(matched);
    }
    return result;
}

fn battleships2(guesses: Vec<Pair<Secret<int>>>,
                ships: Vec<Pair<Secret<int>>>) -> Vec<Secret<boolean>> {
    let mut result = Vec::new();
    for gx, gy in guesses {
        let matched = 1;
        for lx, ly in ships
            matched *= (gx - lx) + (gy - ly);
        result.push(matched == 0);
    }
    return result;
}

fn battleships3(guesses: Matrix<Secret<bool>>,
                ships: Matrix<Secret<bool>>) -> Matrix<Secret<bool>> {
    let mut matrix = matrix::new();
    for i in 0..BOARD_DIMENSION {
        for j in 0..BOARD_DIMENSION
            matrix[i][j] = guesses[i][j] && ships[i][j];
    }
    return matrix;
}

```

Figure 4-1: Three Secure Battleships Implementation in Oblivious Rust. Syntax is lightly edited for readability.

compute the L1 norm of the locations distance, which is 0 only if the ship and guess are equal. To avoid leaking information about the distance between wrong guesses and ship locations, we clamp the product of L1 norms to $[0, 1]$ with a single equality check. As a result, the total bandwidth now improves to $O(|ships| \times b + b^2)$. However, the multiplications in each iteration cannot be executed concurrently, and thus the round complexity is $|ships| + b - 1$. Note that developers can improve this to $O(\log(|ships|) + b - 1)$ by structuring the multiplication as a tree rather than sequentially.

An alternative refinement, shown in `battleships3` requires only a single round overall by using a sparse matrix representation identical to the board in size. A cell in the matrix is set to 1 only if it corresponds to a ship or a guess. A side effect of this is that b can be as small as 1. As a result, the total communication exhibited here is N^2 .

We observe that the given implementations offer us a trade-off between optimizing rounds vs total bandwidth. The last implementation seems incomparable to the others, as it is the only one to depend on N (explicitly), and it does not depend on $|ships|$. However, these parameters are correlated in practice. We know $b \geq \log(N)$, and the number of ships (and guesses) usually depends on the board size (e.g. too many ships can make the game too easy). When looking at a batch of guesses, `battleship3` becomes more attractive as both its network rounds and bandwidth are independent of the number of guesses. More realistic programs exhibit further complex interactions between different parameters. Furthermore, we have not considered the choice of underlying protocols and setup which increases the size of the space.

We show the output of our in-progress resource estimation tool, Carousels, when run on these implementations comparatively in Figure 4.2. We only show *online* network rounds and bandwidth. Carousels also produces plots for other resources including pre-processing. In these plots, we interactively select $b = \lceil \log(N) \rceil$ and

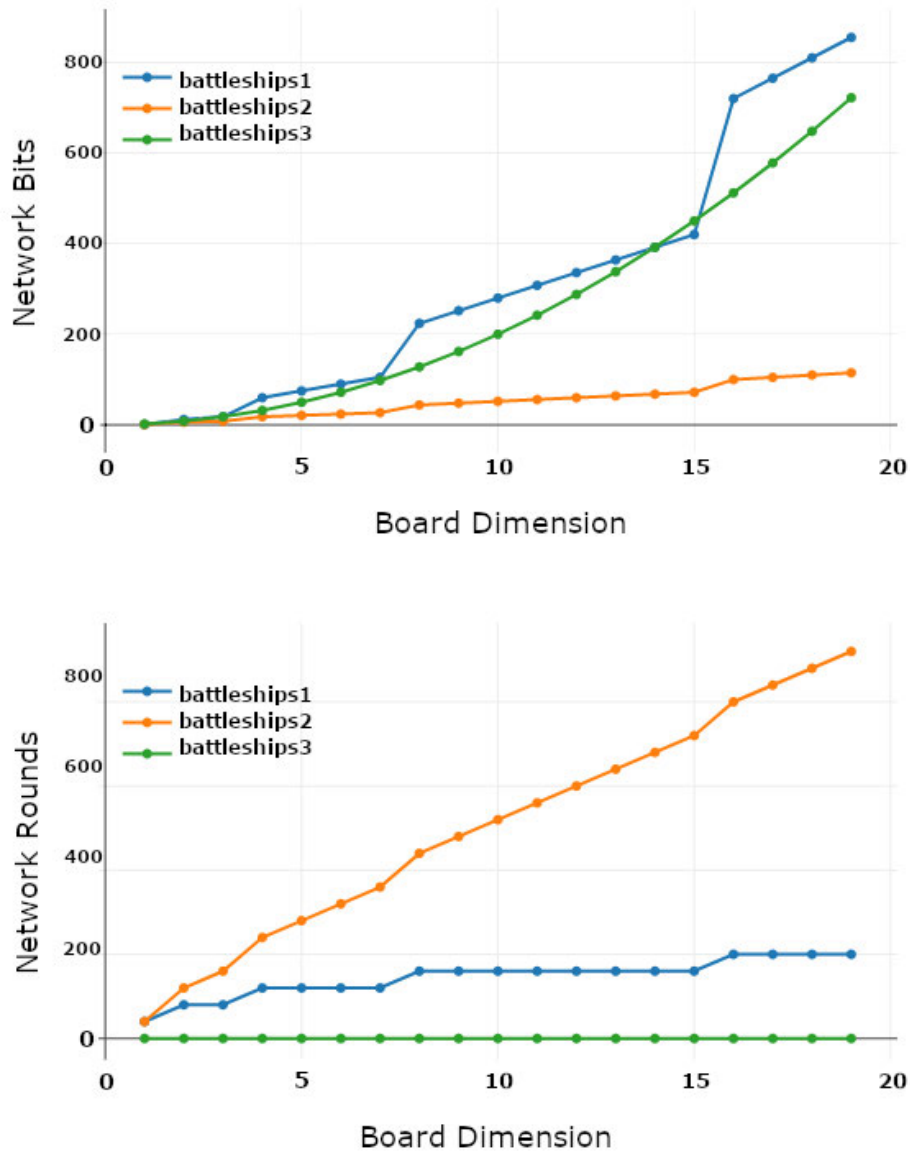


Figure 4:2: The online network bandwidth (top, in bits) and rounds (bottom) for the three battleship implementations.

$|ships| = N$, with only a single guess. In both plots, lower is better. We confirmed the accuracy of these predictions experimentally. Round estimates are exact, and total bandwidth is a close approximation that accounts for all network traffic except metadata packet headers set by the network stack, which are omitted by the underlying cost model.

These factors demonstrate a clear need for developing tools for resource estimation and space exploration. However, toolkit support for secure computation remains thin. Existing frameworks attempt to simplify development process using a variety of embedded programming abstractions [54, 93], or heuristic optimizations (e.g. via domain-specific rewriting rules [245]). Recent toolchains support protocol-agnostic programming [41], provide (slow and at times inaccurate) emulators for performance [54], allows developers to specify security policies for compilers that synthesize distributed programs [19]. In the end, developers are largely left on their own when it comes to analyzing the costs of their programs or attempting ad-hoc optimizations, while also concurrently reasoning about the implications of any potential changes on the correctness, security, and configurations of their programs.

4.3.3 Carousels: Our results

We are currently working on an interactive resource estimation tool for secure computations called Carousels. We are designing Carousels with developers in mind so that they can use it to analyze secure programs and plot concrete resource costs for user-specified regions of the configuration space. Carousels can be used to compare the resource cost of alternative implementations, or explore the protocol and configurations space for a given implementation. We envision Carousels to eventually be used by compilers to guide various heuristic and re-writing based optimizations.

Carousels is framework and protocol agnostic. It operates on a generic intermediate representation; input programs can be transformed into this representation using

pluggable front-end parsers. Carousels’ type system is extensible with external JSON configurations to support language-specific idiosyncrasies and external library calls. At the heart of Carousels are two resource estimation program logics, parameterized over an external cost model. These components allow us to transform programs into a system of symbolic cost recurrences, which Carousels interactively evaluates to produce estimates and plots for user-provided regions of the design space.

Carousels is inspired by existing work on deriving and analyzing cost recurrences [116,160]. However, it relies on several *key design decisions* specific to secure computation. One *key takeaway* is that the fidelity and success of static resource estimation can significantly improve when designs are specialized to a single application domain. Our design is geared towards common secure programs. These are usually perfectly secure, and thus completely oblivious: their resource costs are completely determined by their public configurations and input sizes, and not by their actual secret inputs. It is tractable to “lift” this constant set of relevant parameters, as well as dynamic program flow over them, into our static analysis which we carry out using a dependent type system and a novel resource logic. In rare cases, secure programs may explicitly leak intermediate values to save costs, causing the resource use to depend on leaked information in complex ways. In this case, Carousels follows a best-effort approach and attempts to provide upper bound costs, potentially with user assistance.

Carousels is still work in progress and there are many interesting directions for future work. One possible next step involves performing a larger scale human computer interaction study to determine the usability and effectiveness of our tool among developers and identify key aspects that we can modify and improve. We believe that this would be necessary for the success of any MPC development tool chain and thus MPC itself.

Chapter 5

Conclusion

In this work, I have discussed some of the ways with which we can transition cryptographic tools into practice and bring them to a wider audience.

In Chapter 2 I presented Hecate, a new abuse reporting protocol for end-to-end encrypted messaging systems. With Hecate I showed how emerging questions of accountability do not require the elimination of any of people’s existing privacy guarantees of their secure communication. Hecate is a new construction for asymmetric message franking that is more efficient and more secure than prior works on the topic. Hecate’s main insight is to introduce a pre-processing stage where a moderator can handout batches of tokens to users bound to their identities. These tokens serve a dual purpose: (1) they only allow the moderator to trace the origin of a message and provide confidentiality with respect to everyone else, (2) they allow users to deny having sent any particular message to all non-moderating parties.

In Chapter 3 I presented DP-PIR, a new batched differentially private information retrieval protocol. With DP-PIR I showed one example of practical questions that had until now been completely overlooked in cryptography, and presented an extremely efficient construction that addresses this research gap. DP-PIR is a protocol specifically designed for use cases where the query rate far exceeds the size of the PIR database. In DP-PIR, we amortize the communication and computation work (in the size of the database) of servers down to a constant by relaxing the security guarantees to hold for differential private leakage on the access patterns of the un-

derlying database. Our system batches queries from many users and securely mixes them with dummy queries coming from several servers in order to break the user to query association. We additionally offload expensive public key operations to an offline stage where servers receive the necessary material from users to perform the online stage.

In Chapter 4, I went over some of my work and experience in bringing MPC, and more broadly cryptography, to a wider audience. I believe that the primary reason that these projects succeeded can be attributed to how we directly involved the people in question in the design and development process of our tools. All of these projects focused on understanding the needs and available resources of the people involved. Our works ensured that the end tools are usable, accessible and explainable to a non-technical audience. I then ended the chapter by presenting an ongoing project that I am currently working on that directly builds on this experience with this time developers with no cryptographic background in mind. To that end, I presented Carousels a resource estimation tool-chain that helps developers navigate the large space and many dimensions of MPC protocols.

The common thread among all my work remains the same: refocusing the cryptographic lens on the most important party in any computation, people. In my thesis I demonstrated how by carefully considering and tailoring the various levels of cryptographic work (i.e at the level of formalisms, protocols and software) to suit the various settings and needs of communities at large we can further the adoption of privacy tools in practice. I envision my future collaborations to continue this line of work and include people with even more diverse backgrounds and expertise. I hope that my work and the work of others will eventually show that a road towards hampering surveillance capitalism exists and it starts with ensuring that more people have access to privacy as a first order right.

Bibliography

- [1] 2020 decennial census: Processing the count: Disclosure avoidance modernization. <https://www.census.gov/programs-surveys/decennial-census/decade/2020/planning-management/process/disclosure-avoidance.html>. Accessed: 2022-06-17.
- [2] A federal government privacy-preserving technology demonstration. <https://mccourt.georgetown.edu/news/a-federal-government-privacy-preserving-technology-demonstration/>. Accessed: 2021-10-25.
- [3] Fighting child sexual abuse: Commission proposes new rules to protect children. https://ec.europa.eu/commission/presscorner/detail/en/IP_22_2976. Accessed: 2022-06-17.
- [4] H.r.4174 - foundations for evidence-based policymaking act of 2018. <https://www.congress.gov/bill/115th-congress/house-bill/4174>. Accessed: 2022-06-17.
- [5] match-compute. <https://github.com/Ra1issa/match-compute>. Accessed: 2021-10-28.
- [6] Policy report: Sharing sensitive department of education data across organizational boundaries using secure multiparty computation. https://drive.google.com/file/d/1CURfl3q8j_NOBiaOuPEleJBZFpwQcwti/view. Accessed: 2021-10-25.
- [7] S.3952 - student right to know before you go act of 2022. <https://www.congress.gov/bill/117th-congress/senate-bill/3952?r=3&s=1>. Accessed: 2022-06-17.
- [8] Technical report: Sharing sensitive department of education data across organizational boundaries using secure multiparty computation. https://drive.google.com/file/d/1XyRIgK0dICN9Oxgm2_etCmy4_H0yFnl/view. Accessed: 2021-10-25.
- [9] The promise of evidence-based policymaking. report of the commission on evidence-based policymaking. <https://bipartisanpolicy.org/download/?file=/wp-content/uploads/2019/03/>

Full-Report-The-Promise-of-Evidence-Based-Policymaking-Report-of-the-Comission-on-Evid
pdf. Accessed: 2022-07-10.

- [10] To save everything, click here: Technology, solutionism and the urge to fix problems that don't exist by e. morozov, ed.,. *Info. Pol.*, 18(3):275–276, jul 2013.
- [11] Evidence-based policymaking commission act of 2016, 2016. <https://www.congress.gov/bill/114th-congress/house-bill/1831/text>.
- [12] The promise of evidence-based policymaking: Report of the commission on evidence-based policymaking, 2017. <https://www2.census.gov/adrm/fesac/2017-12-15/Abraham-CEP-final-report.pdf>.
- [13] Batched differentially private information retrieval. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.
- [14] Hecate: Abuse reporting in secure messengers with sealed sender. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.
- [15] Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust encryption. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 480–497. Springer, Heidelberg, February 2010.
- [16] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. Comparing the usability of cryptographic apis. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 154–171, 2017.
- [17] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, 2016.
- [18] Yasemin Acar, Sascha Fahl, and Michelle L Mazurek. You are not your developer, either: A research agenda for usable security and privacy research beyond end users. *2016 IEEE Cybersecurity Development (SecDev)*, pages 3–8, 2016.
- [19] Coc,ku Acay, Rolph Recto, Joshua Gancher, Andrew C Myers, and Elaine Shi. Viaduct: an extensible, optimizing compiler for secure distributed programs. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 740–755, 2021.

- [20] ACDEB. Advisory committee on data for evidence building: Year 1 report. <https://www.bea.gov/system/files/2021-10/acdeb-year-1-report.pdf>, 2021.
- [21] Surabhi Agarwal. India proposes alpha-numeric hash to track WhatsApp chat. *India Times*, 2021.
- [22] Divesh Aggarwal, Ivan Damgr^ard, Jesper Buus Nielsen, Maciej Obremski, Erick Purwanto, João Ribeiro, and Mark Simkin. Stronger leakage-resilient and non-malleable secret sharing schemes for general access structures. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 510–539, Cham, 2019. Springer International Publishing.
- [23] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies*, 2016(2):155–174, 2016.
- [24] Kinan Dak Albab, Rawane Issa, Andrei Lapets, Azer Bestavros, and Nikolaj Volgushev. Scalable secure multi-party network vulnerability analysis via symbolic optimization. In *2017 IEEE Security and Privacy Workshops (SPW)*, pages 211–216, 2017.
- [25] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. MCMix: Anonymous messaging via secure multiparty computation. In Engin Kirda and Thomas Ristenpart, editors, *USENIX Security 2017*, pages 1217–1234. USENIX Association, August 2017.
- [26] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. MCMix: Anonymous messaging via secure multiparty computation. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1217–1234, 2017.
- [27] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 129–158. Springer, Heidelberg, May 2019.
- [28] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277. Springer, Heidelberg, August 2020.
- [29] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 261–290. Springer, Heidelberg, November 2020.

- [30] Abdelrahman Aly, K Cong, D Cozzo, M Keller, E Orsini, D Rotaru, O Scherer, P Scholl, N Smart, T Tanguy, et al. Scale-mamba v1. 12: Documentation, 2021.
- [31] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *IEEE Symposium on Security and Privacy*, pages 962–979. IEEE Computer Society, 2018.
- [32] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 551–569, Savannah, GA, November 2016. USENIX Association.
- [33] Apache. Apache Ignite. <https://github.com/apache/ignite>. Accessed: 2020-12-01.
- [34] Apple. Csam detection technical summary. Accessed: 2022-06-17.
- [35] Dave Archer. Inter-organization sharing of sensitive data for statistics via secure multi-party computation. <https://github.com/GaloisInc/swanky>.
- [36] David W. Archer, Dan Bogdanov, Benny Pinkas, and Pille Pullonen. Maturity and performance of programmable secure computation. *IEEE Security & Privacy*, 14(5):48–56, 2016.
- [37] Saikrishna Badrinarayanan and Akshayaram Srinivasan. Revisiting non-malleable secret sharing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 593–622. Springer, 2019.
- [38] Dirk Balfanz, Glenn Durfee, Rebecca E Grinter, Diana K Smetters, and Paul Stewart. Network-in-a-box: How to set up a secure wireless network in under a minute. In *USENIX Security Symposium*, volume 207, page 222, 2004.
- [39] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. Garbled neural networks are practical. *Cryptology ePrint Archive*, Paper 2019/338, 2019. <https://eprint.iacr.org/2019/338>.
- [40] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. *Cryptology ePrint Archive*, Paper 2016/969, 2016. <https://eprint.iacr.org/2016/969>.
- [41] Yuyan Bao, Kirshanthan Sundararajah, Raghav Malik, Qianchuan Ye, Christopher Wagner, Nouraldin Jaber, Fei Wang, Mohammad Hassan Ameri, Donghang Lu, Alexander Seto, et al. Haccl: An ecosystem for building secure multi-party computations. *arXiv preprint arXiv:2009.01489*, 2020.

- [42] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.
- [43] Amos Beimel and Yuval Ishai. Information-theoretic private information retrieval: A unified construction. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming*, pages 912–926, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- [44] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: Pir with preprocessing. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, page 55–73, Berlin, Heidelberg, 2000. Springer-Verlag.
- [45] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *Journal of Cryptology*, 17(2):125–151, March 2004.
- [46] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *Journal of Cryptology*, 17(2):125–151, 2004.
- [47] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igor Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 619–650. Springer, Heidelberg, August 2017.
- [48] Luca Belli. Whatsapp skewed brazilian election, proving social media's danger to democracy. <https://theconversation.com/whatsapp-skewed-brazilian-election-proving-social-medias-danger-to-democracy-106476>, 2018.
- [49] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 1–10, New York, NY, USA, 1988. Association for Computing Machinery.
- [50] Oliver Berthold and Heinrich Langos. Dummy traffic against long term intersection attacks. In Roger Dingledine and Paul Syverson, editors, *Privacy Enhancing Technologies*, pages 110–128, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- [51] Abhishek Bhowmick, Dan Boneh, Steve Myers, Kunal Talwar, and Karl Tarbe. The Apple PSI system. https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf, 2021.

- [52] Alexander Bienstock, Yevgeniy Dodis, and Paul Rösler. On the price of concurrency in group ratcheting protocols. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 198–228. Springer, Heidelberg, November 2020.
- [53] Alexander Bienstock, Jaiden Fairuze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. A more complete analysis of the signal double ratchet algorithm. Cryptology ePrint Archive, Report 2022/355, 2022. <https://eprint.iacr.org/2022/355>.
- [54] Dan Bogdanov, Sven Laur, and Jan Willemsen. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier Lopez, editors, *Computer Security - ESORICS 2008*, pages 192–206, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [55] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, pages 325–343, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [56] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *IEEE Symposium on Security and Privacy*, pages 762–776. IEEE, 2021.
- [57] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In *Public Key Cryptography (2)*, volume 10175 of *Lecture Notes in Computer Science*, pages 494–524. Springer, 2017.
- [58] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.
- [59] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. <https://toc.cryptobook.us/book.pdf>, 2020.
- [60] Nikita Borisov, George Danezis, and Ian Goldberg. DP5: a private presence service. *Proceedings on Privacy Enhancing Technologies*, 2015(2):4–24, 2015.
- [61] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. Off-the-record communication, or, why not to use PGP. In *WPES*, pages 77–84. ACM, 2004.
- [62] Colin Boyd, Anish Mathuria, and Douglas Stebila. *Protocols for Authentication and Key Establishment, Second Edition*. Information Security and Cryptography. Springer, 2020.

- [63] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, page 1292–1303, New York, NY, USA, 2016. Association for Computing Machinery.
- [64] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In *Theory of Cryptography Conference*, pages 662–693. Springer, 2017.
- [65] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ITCS '12*, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.
- [66] Brazilian fake news draft bill no. 2.630, of 2020. <https://docs.google.com/document/d/1MHMDHsVJBi45PI1R5lAyoLmZvZk8eULHisYFqGy9X2s>, 2020.
- [67] Prasad Buddharapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. Private matching for compute. Cryptology ePrint Archive, Paper 2020/599, 2020. <https://eprint.iacr.org/2020/599>.
- [68] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 1999.
- [69] Sébastien Champion, Julien Devigne, Céline Duguey, and Pierre-Alain Fouque. Multi-device for signal. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *ACNS 20, Part II*, volume 12147 of *LNCS*, pages 167–187. Springer, Heidelberg, October 2020.
- [70] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 694–726, Cham, 2017. Springer International Publishing.
- [71] Ran Canetti, Palak Jain, Marika Swanberg, and Mayank Varia. Universally composable end-to-end secure messaging. Cryptology ePrint Archive, Report 2022/376, 2022. <https://eprint.iacr.org/2022/376>.
- [72] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001.

- [73] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global PKI. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 265–296. Springer, Heidelberg, March 2016.
- [74] Fiona Carroll. *Usable Security and Aesthetics: Designing for Engaging Online Security Warnings and Cautions to Optimise User Security Whilst Affording Ease of Use*, page 23–28. Association for Computing Machinery, New York, NY, USA, 2021.
- [75] Justin Chan, Landon P. Cox, Dean P. Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham M. Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Puneet Sharma, Sudheesh Singanamalla, Jacob E. Sunshine, and Stefano Tessaro. PACT: privacy-sensitive protocols and mechanisms for mobile contact tracing. *IEEE Data Eng. Bull.*, 43(2):15–35, 2020.
- [76] T.-H. Hubert Chan, Kai-Min Chung, Bruce M. Maggs, and Elaine Shi. Foundations of differentially oblivious algorithms. In *SODA*, pages 2448–2467. SIAM, 2019.
- [77] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1445–1459. ACM Press, November 2020.
- [78] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.
- [79] Guoxing Chen, Ten-Hwang Lai, Michael K. Reiter, and Yinqian Zhang. Differentially private access patterns for searchable symmetric encryption. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 810–818, 2018.
- [80] Raymond Cheng, William Scott, Elisaweta Masserova, Irene Zhang, Vipul Goyal, Thomas Anderson, Arvind Krishnamurthy, and Bryan Parno. Talek: Private group messaging with hidden access patterns, 2020.
- [81] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 375–403, Cham, 2019. Springer International Publishing.
- [82] Benny Chor and Niv Gilboa. Computationally private information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 304–313, 1997.

- [83] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, nov 1998.
- [84] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, November 1998.
- [85] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. *Journal of Cryptology*, 33(4):1914–1983, 2020.
- [86] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1802–1819. ACM Press, October 2018.
- [87] Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On post-compromise security. In *CSF*, pages 164–178. IEEE Computer Society, 2016.
- [88] Nicholas Confessore. Cambridge analytica and facebook: The scandal and the fallout so far. Accessed: 2022-06-17.
- [89] Henry Corrigan-Gibbs. Privacy-preserving telemetry in Firefox. Real World Crypto (RWC), 2020. <https://rwc.iacr.org/2020/slides/Gibbs.pdf>.
- [90] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE Computer Society Press, May 2015.
- [91] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 44–75, Cham, 2020. Springer International Publishing.
- [92] Alex Cranz. There are over 3 billion active Android devices. <https://www.theverge.com/2021/5/18/22440813/android-devices-active-number-smartphones-google-2021>. Accessed: 2021-06-02.
- [93] Kinan Dak Albab, Rawane Issa, Andrei Lapets, Peter Flockhart, Lucy Qin, and Ira Globus-Harris. Tutorial: Deploying secure multi-party computation on the web using jiff. In *2019 IEEE Cybersecurity Development (SecDev)*, pages 3–3, 2019.

- [94] dalek-cryptography. ed25519-dalek.
<https://github.com/dalek-cryptography/ed25519-dalek>, 2020.
- [95] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 643–662, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- [96] Darren Davis, Fabian Monrose, and Michael K Reiter. On user choice in graphical password schemes. In *USENIX security symposium*, volume 13, pages 11–11, 2004.
- [97] David Adrian Deirdre Connolly, Thomas Ptacek and Matthew D. Green. Apple’s csam detection, feat. matthew green. Accessed: 2022-06-17.
- [98] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.
- [99] Frank Denis. The sodium cryptography library.
<https://download.libsodium.org/doc/>, Jun 2013.
- [100] Rachna Dhamija and Adrian Perrig. Deja {Vu–A} user study: Using images for authentication. In *9th USENIX Security Symposium (USENIX Security 00)*, 2000.
- [101] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *USENIX Security 2004*, pages 303–320. USENIX Association, August 2004.
- [102] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 155–186. Springer, Heidelberg, August 2018.
- [103] Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. Composability and on-line deniability of authentication. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 146–162. Springer, Heidelberg, March 2009.
- [104] DP-PIR GitHub repository.
<https://github.com/multiparty/DP-PIR/tree/usenix2022>, 2022.
- [105] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

- [106] Elizabeth Dwoskin and Annie Gowen. On WhatsApp, fake news is fast – and can be fatal. https://www.washingtonpost.com/business/economy/on-whatsapp-fake-news-is-fast--and-can-be-fatal/2018/07/23/a2dd7112-8ebf-11e8-bcd5-9d911c784c38_story.html, July 2018. Accessed: 09-28-2020.
- [107] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *SODA*, pages 2468–2479. SIAM, 2019.
- [108] Ethereum. Ethereum: Privacy. <https://docs.ethhub.io/ethereum-roadmap/privacy/>. Accessed: 2022-06-17.
- [109] Facebook. The value of secure multi-party computation. <https://privacytech.fb.com/multi-party-computation/>. Accessed: 2022-06-17.
- [110] Facebook. Messenger secret conversations: Technical whitepaper (version 2.0). <https://about.fb.com/wp-content/uploads/2016/07/messenger-secret-conversations-technical-whitepaper.pdf>, 2017.
- [111] Pooya Farshim, Benoît Libert, Kenneth G. Paterson, and Elizabeth A. Quaglia. Robust encryption, revisited. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 352–368. Springer, Heidelberg, February / March 2013.
- [112] Pooya Farshim, Claudio Orlandi, and Răzvan Rocșie. Security of symmetric primitives under incorrect usage of keys. *IACR Trans. Symm. Cryptol.*, 2017(1):449–473, 2017.
- [113] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy&paste on android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 121–136, 2017.
- [114] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 60–77. Springer, Heidelberg, August 2006.
- [115] Marc Fischlin and Dominique Schröder. On the impossibility of three-move blind signature schemes. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 197–215. Springer, Heidelberg, May / June 2010.
- [116] Antonio Flores-Montoya and Reiner Hähnle. Resource analysis of complex programs with cost equations. In Jacques Garrigue, editor, *Programming*

- Languages and Systems*, pages 275–295, Cham, 2014. Springer International Publishing.
- [117] Nationale Center for Education Statistics. 2015–16 national postsecondary student aid study (npsas:16) student financial aid estimates for 2015–16, 2016. <https://nces.ed.gov/pubs2018/2018466.pdf>.
- [118] Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel Weitzner. Practical accountability of secret processes. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 657–674, Baltimore, MD, August 2018. USENIX Association.
- [119] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS '02*, page 193–206, New York, NY, USA, 2002. Association for Computing Machinery.
- [120] Georg Fuchsbauer, Christian Hanser, Chethan Kamath, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model from weaker assumptions. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 391–408. Springer, Heidelberg, August / September 2016.
- [121] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 233–253. Springer, Heidelberg, August 2015.
- [122] Inc. Galois. swanky: A suite of rust libraries for secure multi-party computation. <https://github.com/GaloisInc/swanky>.
- [123] Simson L Garfinkel and Robert C Miller. Johnny 2: a user test of key continuity management with s/mime and outlook express. In *Proceedings of the 2005 symposium on Usable privacy and security*, pages 13–24, 2005.
- [124] Oded Goldreich. Towards a theory of software protection and simulation by Oblivious RAMs. In *STOC*, pages 182–194. ACM, 1987.
- [125] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on Oblivious RAMs. *J. ACM*, 43(3):431–473, May 1996.
- [126] Nathaniel S Good and Aaron Krekelberg. Usability and privacy: a study of kazaa p2p file-sharing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 137–144, 2003.

- [127] Vipul Goyal and Ashutosh Kumar. Non-malleable secret sharing. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 685–698, 2018.
- [128] Sen. Lindsey Graham, Sen. Richard Blumenthal, Sen. Kevin Cramer, Sen. Dianne Feinstein, Sen. Josh Hawley, Sen. Doug Jones, Sen. Robert Casey, Sen. Sheldon Whitehouse, Sen. Richard Durbin, Sen. Joni Ernst, Sen. John Kennedy, Sen. Ted Cruz, Sen. Chuck Grassley, Sen. Rob Portman, Sen. Lisa Murkowski, Sen. John Cornyn, and Sen. Kelly Loeffler. S.3398 - EARN IT act of 2020, 2020.
- [129] Matthew Green, Watson Ladd, and Ian Miers. A protocol for privately reporting ad impressions at scale. In *CCS*, pages 1591–1601. ACM, 2016.
- [130] Matthew D. Green. Earn it is a direct attack on end-to-end encryption. Accessed: 2022-06-17.
- [131] Matthew D. Green and Alex Stamos. Apple wants to protect children. but it's creating serious privacy risks. Accessed: 2022-06-17.
- [132] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 66–97. Springer, Heidelberg, August 2017.
- [133] Jan Gugenheimer, Wen-Jie Tseng, Abraham Hani Mhaidli, Jan Ole Rixen, Mark McGill, Michael Nebeling, Mohamed Khamis, Florian Schaub, and Sanchari Das. Novel challenges of safety, security and privacy in extended reality. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI EA '22, New York, NY, USA, 2022. Association for Computing Machinery.
- [134] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 29–37. Springer, Heidelberg, April 1990.
- [135] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with Popcorn. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 91–107, Santa Clara, CA, March 2016. USENIX Association.
- [136] Marco Gutfleisch, Jan H Klemmer, Niklas Busch, Yasemin Acar, M Angela Sasse, and Sascha Fahl. How does usable security (not) end up in software products? results from a qualitative interview study. In *43rd IEEE Symposium on Security and Privacy, IEEE S&P*, pages 22–26, 2022.

- [137] Ariel Hamlin, Rafail Ostrovsky, Mor Weiss, and Daniel Wichs. Private anonymous data access. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 244–273, Cham, 2019. Springer International Publishing.
- [138] Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. Sok: General purpose compilers for secure multi-party computation. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1220–1237, 2019.
- [139] Marcella Christine Hastings. *Secure Multi-Party Computation in Practice*. PhD thesis, University of Pennsylvania, 2021.
- [140] Hecate Rust implementation. <https://github.com/Ra1issa/hecate>, 2022.
- [141] Hecate’s modified libsignal-client implementation. <https://github.com/Ra1issa/libsignal-client/>, 2022.
- [142] Hecate’s modified signal-cli implementation. <https://github.com/Ra1issa/signal-cli>, 2022.
- [143] Mark Hosenball. Factbox: Key findings from senate inquiry into russian interference in 2016 u.s. election. Accessed: 2022-06-17.
- [144] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. Cryptology ePrint Archive, Paper 2019/723, 2019. <https://eprint.iacr.org/2019/723>.
- [145] Mansoor Iqbal. App download and usage statistics (2020). <https://www.businessofapps.com/data/app-statistics/>. Accessed: 2021-06-02.
- [146] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, STOC ’04*, page 262–271, New York, NY, USA, 2004. Association for Computing Machinery.
- [147] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from anonymity. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS’06)*, pages 239–248, 2006.
- [148] Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

- [149] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 143–154. Springer, Heidelberg, May 1996.
- [150] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 159–188. Springer, Heidelberg, May 2019.
- [151] Daniel Jost, Ueli Maurer, and Marta Mularczyk. A unified and composable take on ratcheting. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 180–210. Springer, Heidelberg, December 2019.
- [152] Daniel Kales. Golang DPF library. <https://github.com/dkales/dpf-go>, 2021.
- [153] Seny Kamara. Crypto for the people. Invited Talk, 2020.
- [154] Seny Kamara, Mallory Knodel, Emma Llansó, Greg Nojeim, Lucy Qin, Dhanaraj Thakur, and Caitlin Vogus. Outside looking in: Approaches to content moderation in end-to-end encrypted systems. <https://cdt.org/wp-content/uploads/2021/08/CDT-Outside-Looking-In-Approaches-to-Content-Moderation-in-End-to-End-Encrypted-Systems.pdf>, 2021.
- [155] Seny Kamara, Tarik Moataz, Andrew Park, and Lucy Qin. A decentralized and encrypted national gun registry. Cryptology ePrint Archive, Report 2021/107, 2021. <https://ia.cr/2021/107>.
- [156] Gabriel Kaptchuk. Weaving social accountability into cryptographic protocols (charles river area crypto day 2022), 2022.
- [157] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Round-optimal blind signatures in the plain model from classical and quantum standard assumptions. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 404–434. Springer, Heidelberg, October 2021.
- [158] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.
- [159] Stephen T Kent. Internet privacy enhanced mail. *Communications of the ACM*, 36(8):48–60, 1993.

- [160] Zachary Kincaid, Jason Breck, Ashkan Forouhi Boroujeni, and Thomas Reps. Compositional recurrence analysis revisited. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, page 248–262, New York, NY, USA, 2017. Association for Computing Machinery.
- [161] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *arXiv preprint arXiv:2109.00984*, 2021.
- [162] Dmitry Kogan and Henry Corrigan-Gibbs. Private blocklist lookups with checklist. In *USENIX Security Symposium*, pages 875–892. USENIX Association, 2021.
- [163] Anunay Kulshrestha and Jonathan Mayer. Identifying harmful media in end-to-end encrypted communication: Efficient private membership computation. In *USENIX Security Symposium*. USENIX Association, 2021.
- [164] Anunay Kulshrestha and Jonathan R. Mayer. Identifying harmful media in end-to-end encrypted communication: Efficient private membership computation. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 893–910. USENIX Association, August 2021.
- [165] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, page 364, USA, 1997. IEEE Computer Society.
- [166] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: An efficient communication system with strong anonymity. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115–134, 2016.
- [167] Michael Labib. Turbocharge Amazon S3 with Amazon ElastiCache for Redis. <https://aws.amazon.com/blogs/storage/turbocharge-amazon-s3-with-amazon-elasticache-for-redis/>. Accessed: 2020-12-01.
- [168] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16. Springer, Heidelberg, November 2007.
- [169] Andrei Lapets, Kinan Dak Albab, Rawane Issa, Lucy Qin, Mayank Varia, Azer Bestavros, and Frederick Jansen. Role-based ecosystem for the design, development, and deployment of secure multi-party data analytics

- applications. In *2019 IEEE Cybersecurity Development (SecDev)*, pages 129–140, 2019.
- [170] Andrei Lapets, Frederick Jansen, Kinan Dak Albab, Rawane Issa, Lucy Qin, Mayank Varia, and Azer Bestavros. Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies, COMPASS '18*, New York, NY, USA, 2018. Association for Computing Machinery.
- [171] Andrei Lapets, Frederick Jansen, Kinan Dak Albab, Rawane Issa, Lucy Qin, Mayank Varia, and Azer Bestavros. Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies, COMPASS '18*, New York, NY, USA, 2018. Association for Computing Machinery.
- [172] David Lazar, Yossi Gilad, and Nikolai Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 711–725, 2018.
- [173] David Lazar and Nikolai Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *OSDI*, pages 571–586. USENIX Association, 2016.
- [174] Sherman Lee. Privacy revolution: How blockchain is reshaping our economy. *Forbes*. Accessed: 2022-06-17.
- [175] Iraklis Leontiadis and Serge Vaudenay. Private message franking with after opening privacy. Cryptology ePrint Archive, Report 2018/938, 2018. <https://eprint.iacr.org/2018/938>.
- [176] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *Information Security*, pages 314–328, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- [177] Helger Lipmaa. First CIPR protocol with data-dependent computation. In *Proceedings of the 12th International Conference on Information Security and Cryptology, ICISC'09*, page 193–210, Berlin, Heidelberg, 2009. Springer-Verlag.
- [178] Linsheng Liu, Daniel S. Roche, Austin Theriault, and Arkady Yerukhimovich. Fighting fake news in encrypted messaging with the fuzzy anonymous complaint tally system (FACTS). In *NDSS*. The Internet Society, 2022.

- [179] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security*, pages 168–186, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- [180] Tao Luo, Mingen Pan, Pierre Tholoniati, Asaf Cidon, Roxana Geambasu, and Mathias Lécuyer. Privacy budget scheduling. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 55–74, 2021.
- [181] Alexandra Ma and Ben Gilbert. Facebook understood how dangerous the trump-linked data firm cambridge analytica could be much earlier than it previously said. here’s everything that’s happened up until now. Accessed: 2022-06-17.
- [182] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schröder. Maliciously secure multi-client ORAM. In *International Conference on Applied Cryptography and Network Security*, pages 645–664. Springer, 2017.
- [183] Manish Singh. Whatsapp is now delivering roughly 100 billion messages a day. <https://techcrunch.com/2020/10/29/whatsapp-is-now-delivering-roughly-100-billion-messages-a-day/>, 2020.
- [184] Ian Martiny, Gabriel Kaptchuk, Adam Aviv, Dan Roche, and Eric Wustrow. Improving Signal’s sealed sender. In *NDSS*. The Internet Society, 2021.
- [185] Nick Mathewson and Roger Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In David Martin and Andrei Serjantov, editors, *Privacy Enhancing Technologies*, pages 17–34, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- [186] Sahar Mazloom and S. Dov Gordon. Secure computation with differentially private access patterns. In *CCS*, pages 490–507. ACM, 2018.
- [187] Microsoft. <https://www.microsoft.com/en-us/research/project/microsoft-seal/>, 2017. Online. Accessed on 15-Nov-2021.
- [188] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using one-sided RDMA reads to build a fast, CPU-efficient key-value store. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 103–114, 2013.
- [189] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. In *Proceedings of the 20th USENIX Conference on Security, SEC’11*, page 31, USA, 2011. USENIX Association.

- [190] Payman Mohassel and Peter Rindal. Aby^3 : A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 35–52, New York, NY, USA, 2018. Association for Computing Machinery.
- [191] moxie0. Advanced cryptographic ratcheting. <https://signal.org/blog/advanced-ratcheting/>.
- [192] MPC Alliance. <https://www.mpcalliance.org/>, 2021.
- [193] Daniela Napoli. Developing accessible and usable security (accus) heuristics. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI EA '18, page 1–6, New York, NY, USA, 2018. Association for Computing Machinery.
- [194] Sarah Kinoshian Nelson Renteria and Rodrigo Campos. Analysis: Crypto crash leaves el salvador with no easy exit from worsening crisis. *Reuters*. Accessed: 2022-06-17.
- [195] Lily Hay Newman. The earn it act is a sneak attack on encryption. Accessed: 2022-06-17.
- [196] Nth Party. <https://www.nthparty.com/>, 2021.
- [197] Femi Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In George Danezis, editor, *Financial Cryptography and Data Security*, pages 158–172, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- [198] Oversight Board. Ensuring respect for free expression, through independent judgment. <https://oversightboard.com/>, 2021.
- [199] Rahul Parhi, Michael Schliep, and Nicholas Hopper. MP3: a more efficient private presence protocol. In Sarah Meiklejohn and Kazuo Sako, editors, *Financial Cryptography and Data Security*, pages 38–57, Berlin, Heidelberg, 2018. Springer Berlin Heidelberg.
- [200] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Private stateful information retrieval. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1002–1019, New York, NY, USA, 2018. Association for Computing Machinery.
- [201] Bryan D. Payne and W. Keith Edwards. A brief introduction to usable security. *IEEE Internet Computing*, 12(3):13–21, may 2008.

- [202] Charlotte Peale, Saba Eskandarian, and Dan Boneh. Secure complaint-enabled source-tracking for encrypted messaging. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1484–1506. ACM Press, November 2021.
- [203] Charlotte Peale, Saba Eskandarian, and Dan Boneh. Secure complaint-enabled source-tracking for encrypted messaging. *CCS '21*, page 1484–1506, New York, NY, USA, 2021. Association for Computing Machinery.
- [204] Jason Perry, Debayan Gupta, Joan Feigenbaum, and Rebecca N. Wright. Systematizing secure computation for research and decision support. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks*, pages 380–397, Cham, 2014. Springer International Publishing.
- [205] Riana Pfefferkorn. Content-oblivious trust and safety techniques: Results from a survey of online service providers. <https://ssrn.com/abstract=3920031>, 2021.
- [206] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, Heidelberg, May 2019.
- [207] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based psi with linear communication. *Cryptology ePrint Archive*, Paper 2019/241, 2019. <https://eprint.iacr.org/2019/241>.
- [208] Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2018.
- [209] Sigmund N. Porter. A password extension for improved human factors. *Comput. Secur.*, 1(1):54–56, 1982.
- [210] Newley Purnell and Jeff Horowitz. WhatsApp says it filed suit in India to prevent tracing of encrypted messages. <https://www.wsj.com/articles/whatsapp-says-it-filed-suit-in-india-to-prevent-tracing-of-encrypted-messages-11622000307>, 2021.
- [211] Lucy Qin, Andrei Lapets, Frederick Jansen, Peter Flockhart, Kinan Dak Albab, Ira Globus-Harris, Shannon Roberts, and Mayank Varia. From usability to secure computing and back again. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 191–210, Santa Clara, CA, August 2019. USENIX Association.

- [212] Anjana Rajan, Lucy Qin, David W Archer, Dan Boneh, Tancrede Lepoint, and Mayank Varia. Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, pages 1–4, 2018.
- [213] Phillip Rogaway. The moral character of cryptographic work. Austin, TX, August 2016. USENIX Association.
- [214] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: On the end-to-end security of group chats in Signal, WhatsApp, and Threema. In *EuroS&P*, pages 415–429. IEEE, 2018.
- [215] Shahrokh Saeednia, Steve Kremer, and Olivier Markowitch. An efficient strong designated verifier signature scheme. In Jong In Lim and Dong Hoon Lee, editors, *ICISC 03*, volume 2971 of *LNCS*, pages 40–54. Springer, Heidelberg, November 2004.
- [216] Sarah Scheffler and Jonathan Mayer. Sok: Content moderation in end to end encryption, 2022.
- [217] Sarah Scheffler and Jonathan Mayer. SoK: content moderation in end-to-end encryption, 2022.
- [218] Michael Schliep and Nicholas Hopper. End-to-end secure mobile group messaging with conversation integrity and deniability. In *WPES@CCS*, pages 55–73. ACM, 2019.
- [219] Leo Schwartz and Abubakar Idris. From argentina to nigeria, people saw terra as more stable than local currency. they lost everything. *rest of world*. Accessed: 2022-06-17.
- [220] Sebastian Scheibner. signal-cli. <https://github.com/AsamK/signal-cli>.
- [221] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [222] Abhi Shelat, Schuyler Rosefield, Jack Doerner, Nathan Lilienthal, and Megan Chen. Obliv rust manuscript. Unpublished, N.D.
- [223] Signal. Technology preview: Sealed sender for signal. <https://signal.org/blog/sealed-sender/>, 2018.
- [224] Signal. libsignal-client. <https://github.com/signalapp/libsignal-client>, 2020.
- [225] Signal. Technical information. <https://signal.org/docs/>, 2021.
- [226] Signal. Signal-server. <https://github.com/signalapp/Signal-Server>, 2022.

- [227] Kate Starbird. Online rumors, misinformation and disinformation: The perfect storm of covid-19 and election2020. In *Enigma 2021*. USENIX Association, February 2021.
- [228] Stephanie Straus. A federal government privacy preserving technology demonstration, 2021. <https://mccourt.georgetown.edu/news/a-federal-government-privacy-preserving-technology-demonstration/>.
- [229] BTC STUDIOS. Particl: A privacy revolution is coming. *Bitcoin Magazine*. Accessed: 2022-06-17.
- [230] Li Q. Tay, Mark J. Hurlstone, Tim Kurz, and Ullrich K. H. Ecker. A comparison of prebunking and debunking interventions for implied versus explicit misinformation. *PsyArXiv*, 2021.
- [231] Kurt Thomas, Devdatta Akhawe, Michael Bailey, Dan Boneh, Elie Bursztein, Sunny Consolvo, Nicola Dell, Zakir Durumeric, Patrick Gage Kelley, Deepak Kumar, Damon McCoy, Sarah Meiklejohn, Thomas Ristenpart, and Gianluca Stringhini. SoK: Hate, harassment, and the changing landscape of online abuse. In *2021 IEEE Symposium on Security and Privacy*, pages 247–267. IEEE Computer Society Press, May 2021.
- [232] Julie Thorpe and Paul C van Oorschot. Graphical dictionaries and the memorable space of graphical passwords. In *USENIX Security Symposium*, pages 135–150, 2004.
- [233] Raphael R Toledo, George Danezis, and Ian Goldberg. Lower-cost ϵ -private information retrieval. *Proceedings on Privacy Enhancing Technologies*, 2016(4):184–201, 2016.
- [234] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy. *IEEE Data Eng. Bull.*, 43(2):95–107, 2020.
- [235] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James R. Larus, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, Ludovic Barman, Sylvain Chatel, Kenneth G. Paterson, Srdjan Capkun, David A. Basin, Jan Beutel, Dennis Jackson, Marc Roeschlin, Patrick Leu, Bart Preneel, Nigel P. Smart, Aysajan Abidin, Seda Gurses, Michael Veale, Cas Cremers, Michael Backes, Nils Ole Tippenhauer, Reuben Binns, Ciro Cattuto, Alain Barrat, Dario Fiore, Manuel Barbosa, Rui Oliveira, and José Pereira. Decentralized privacy-preserving proximity tracing. *IEEE Data Eng. Bull.*, 43(2):36–66, 2020.

- [236] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 423–440, 2017.
- [237] Nirvan Tyagi, Paul Grubbs, Julia Len, Ian Miers, and Thomas Ristenpart. Asymmetric message franking: Content moderation for metadata-private end-to-end encryption. *Cryptology ePrint Archive*, Paper 2019/565, 2019. <https://eprint.iacr.org/2019/565>.
- [238] Nirvan Tyagi, Paul Grubbs, Julia Len, Ian Miers, and Thomas Ristenpart. Asymmetric message franking: Content moderation for metadata-private end-to-end encryption. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 222–250. Springer, Heidelberg, August 2019.
- [239] Nirvan Tyagi, Ian Miers, and Thomas Ristenpart. Traceback for end-to-end encrypted messaging. *Cryptology ePrint Archive*, Paper 2019/981, 2019. <https://eprint.iacr.org/2019/981>.
- [240] Nirvan Tyagi, Ian Miers, and Thomas Ristenpart. Traceback for end-to-end encrypted messaging. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 413–430. ACM Press, November 2019.
- [241] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. SoK: Secure messaging. In *2015 IEEE Symposium on Security and Privacy*, pages 232–249. IEEE Computer Society Press, May 2015.
- [242] Salil Vadhan. The complexity of differential privacy. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pages 347–450. Springer International Publishing, Cham, 2017.
- [243] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP ’15, page 137–152, New York, NY, USA, 2015. Association for Computing Machinery.
- [244] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *SOSP*, pages 137–152. ACM, 2015.

- [245] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. Conclave: Secure multi-party computation on big data. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, New York, NY, USA, 2019. Association for Computing Machinery.
- [246] Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.
- [247] Sameer Wagh, Paul Cuff, and Prateek Mittal. Differentially private Oblivious RAM. *Proceedings on Privacy Enhancing Technologies*, 2018(4):64–84, 2018.
- [248] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 299–313, Boston, MA, March 2017. USENIX Association.
- [249] Xiao Wang, Alex J Malozemoff, and Jonathan Katz. Emp-toolkit: Efficient multiparty computation toolkit, 2016.
- [250] WhatsApp. Two billion users – connecting the world privately. <https://blog.whatsapp.com/two-billion-users-connecting-the-world-privately/>, 2020.
- [251] Alma Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of PGP 5.0. In *8th USENIX Security Symposium (USENIX Security 99)*, Washington, D.C., August 1999. USENIX Association.
- [252] Susan Wiedenbeck, Jim Waters, Jean-Camille Birget, Alex Brodskiy, and Nasir Memon. Authentication using graphical passwords: Effects of tolerance and image choice. In *Proceedings of the 2005 Symposium on Usable Privacy and Security*, SOUPS '05, page 1–12, New York, NY, USA, 2005. Association for Computing Machinery.
- [253] David J. Wu, Joe Zimmerman, J  r  my Planul, and John C. Mitchell. Privacy-preserving shortest path computation. In *NDSS*. The Internet Society, 2016.
- [254] Liang Wu, Fred Morstatter, Kathleen M Carley, and Huan Liu. Misinformation in social media: definition, manipulation, and detection. *ACM SIGKDD Explorations Newsletter*, 21(2):80–90, 2019.
- [255] Jian Yang, Joseph Izraelevitz, and Steven Swanson. Orion: A distributed file system for non-volatile main memory and RDMA-capable networks. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 221–234, 2019.

- [256] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.
- [257] Ka-Ping Yee. User interaction design for secure systems. In *International Conference on Information and Communications Security*, pages 278–290. Springer, 2002.
- [258] Zcash. How zk-snarks are constructed in zcash.
<https://z.cash/technology/zksnarks/>. Accessed: 2022-06-17.
- [259] Shoshana Zuboff. *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power*. 1st edition, 2018.

Curriculum Vitae

