PRACTICE-ORIENTED PRIVACY IN CRYPTOGRAPHY

by
Alishah Chator

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland
March, 2023

Abstract

While formal cryptographic schemes can provide strong privacy guarantees, heuristic schemes that prioritize efficiency over formal rigor are often deployed in practice, which can result in privacy loss. Academic schemes that do receive rigorous attention often lack concrete efficiency or are difficult to implement. This creates tension between practice and research, leading to deployed privacy-preserving systems that are not backed by strong cryptographic guarantees.

To address this tension between practice and research, we propose a practice-oriented privacy approach, which focuses on designing systems with formal privacy models that can effectively map to real-world use cases. This approach includes analyzing existing privacy-preserving systems to measure their privacy guarantees and how they are used. Furthermore, it explores solutions in the literature and analyzes gaps in their models to design augmented systems that apply more clearly to practice.

We focus on two settings of privacy-preserving payments and communications. First, we introduce BlockSci, a software platform that can be used to perform analyses on the privacy and usage of blockchains. Specifically, we assess the privacy of the Dash cryptocurrency and analyze the velocity of cryptocurrencies, finding that Dash's PrivateSend may still be vulnerable to clustering attacks and that a significant fraction of transactions on Bitcoin are "self-churn" transactions.

Next, we build a technique for reducing bandwidth in mixing cryptocurrencies, which suffer from a practical limitation: the size of the transaction growing linearly

with the size of the anonymity set. Our proposed technique efficiently samples cover traffic from a finite and public set of known values, while deriving a compact description of the resulting transaction set. We show how this technique can be integrated with various currencies and different cover sampling distributions.

Finally, we look at the problem of establishing secure communication channels without access to a trusted public key infrastructure. We construct a scheme that uses network latency and reverse Turing tests to detect the presence of eavesdroppers, prove our construction secure, and implement it on top of an existing communication protocol.

This line of work bridges the gap between theoretical cryptographic research and real-world deployments to bring better privacy-preserving schemes to end users.

Thesis Readers

Dr. Matthew Green (Primary Advisor)
Associate Professor
Department of Computer Science
Johns Hopkins University

Dr. Abhishek Jain
Associate Professor
Department of Computer Science
Johns Hopkins University

Dr. Yinzhi Cao
Assistant Professor
Department of Computer Science
Johns Hopkins University

Dedicated to my Mom, Dad, and Areeb.

Acknowledgements

My long, transformative journey as a graduate student was only possible thanks to many, many people. First, I want to thank my advisor Matthew Green. I can't think of a better mentor than Matt for learning how to be a researcher, a teacher, and a privacy advocate. I am so grateful to him for his support, guidance, and encouragement for me to find my own path as a researcher. My fondest memories of my PhD are standing in front of a whiteboard with Matt and discussing everything from research ideas to sci-fi novels. He is truly an inspiration of what an academic can be, with his deep insight into theory, practice, and pretty much anything privacy or technology related. Being his student was an incredible experience, and I hope to continue having opportunities to learn from him.

Next, I would like to thank Abhishek Jain. Abhishek really opened my eyes to the possibilities of cryptography and I learn something every time I speak with him. His systematic and enthusiastic approach to understanding a cryptographic construction really changed the way I read and absorb new research. The range of problems that Abhishek works on is truly remarkable and he always has valuable insight whenever I come to him with a research question.

I would like to thank Yinzhi Cao for really growing the scope of our security and privacy group at Hopkins, and for agreeing to serve on my thesis committee.

I want to thank Susan Hohenberger for mentorship as well. Working with Susan really helped me find my confidence and grounding as a researcher. Her guidance on

formalism and devising research problems was invaluable.

I was fortunate to be hosted for a summer by Arvind Narayanan at Princeton. Working with Arvind, I was able to gain a deeper understanding of privacy research beyond cryptography. Arvind's understanding of technology's impact on society is incredible and I am grateful to him for giving me the opportunity to work with him.

I also got the opportunity to work with Nick Sullivan and the Cryptography Research team at Cloudflare. I never imagined getting an internship through a Twitter DM, but I am so thankful to have gotten the chance to see what industry research, protocol standardization, and deploying privacy systems at scale looks like.

When COVID-19 hit and my summer research plans fell through, Nadia Heninger at UCSD kindly agreed to let me work with her remotely. I am so grateful for both the mentorship during a difficult time and the opportunity to learn more about the area of cryptanalysis.

When I was feeling lost and not sure if privacy research could make a real difference, Seny Kamara provided me with invaluable guidance on how to be an academic without losing touch with the communities we are trying to build better systems for. I am grateful to him, Lucy, Leah, and the rest of the Community-Driven Crypto seminar members for rekindling my passion for research.

I am thankful to Alex Halderman, Zakir Durumeric, and David Adrian for getting me started on my research journey back at Michigan.

I would like to thank all of my incredible collaborators: Steven Goldfeder, Matthew Green, Mathias Hall-Andersen, Harry Kalodner, Kevin Lee, Malte Möser, Arvind Narayanan, Martin Plattner, Nick Sullivan, and David Wong.

I am also grateful to have been surrounded by some wonderful labmates - Arka, Aarushi, Gabe, Dave, Christina, Ian, Gijs, Nils, Zhengzhong, Gabby, Tushar, Max, Pratyush, Harry, Aditya, Logan, David, and Atheer. I am especially lucky to have

started my program at the same time as Aarushi and Arka. Over the years, we have had so many conversations ranging from deeply technical to chaotically absurd. Both of their support through the years has been immeasurable and our friendship is very dear to me. Our lab overall has been such a collaborative and welcoming place and I am thankful to everyone that made it that way.

Within the CS department, I am also grateful to all the friends I met at happy hours, while making coffee, and when just looking for a distraction from work - Yasamin, Aditya, Ama, Jaron, Ravi, Enayat, and Rohit to name a few. I would like to thank the Baltimore Ismaili community, and Saleema and Anita in particular, for helping me get settled and feeling at home here. Baltimore is a delightful city and I am glad I got to spend these years here.

I also have so many friends and family whose unconditional support has allowed me to keep pushing forward, thank you and I love you all deeply. A special thanks goes to Kinari, Abhi, Deepak, and Sidu. Kinari for being such a wonderful roommate and basically my sister, I can't imagine how I would have gotten through grad school without you. Abhi, Deepak, and Sidu were also major sources of encouragement, especially during some of my lowest lows in grad school. I am also grateful to QPL and all my Zoom coworking and game playing friends for making the pandemic feel much less isolating.

I also want to thank Prateek, who left us far too soon, but without whose help I would not know the first thing about grad school. You will alway be in my thoughts.

Finally, I must thank my mom, dad, Areeb, and Nia. My mom and dad always supported me and enabled me to find my own path. Without their love and sacrifices I would not be here. Areeb has always been an incredibly supportive older brother and my first and greatest inspiration. Our beagle, Nia, joined our family during the pandemic and I thank her for being adorable.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

One of the central goals of cryptography is providing privacy. As with many cryptographic properties, the definition of privacy varies, but generally refers to the goal of hiding some information that the user deems secret. This information ranges from the content of a message, to the identity of the sender, or even to a specific input to a protocol. As the nature of what needs to be hidden changes from situation to situation, the result is that custom specific solutions are common in this space. Without general approaches, both protocol designers and deployers have a great deal of leeway in shaping these systems. While flexibility can have its advantages, in security it can also lead to a lack of consistency and compatibility.

For example, deployers of protocols will often be concerned with concrete efficiencies of schemes rather than the formal rigor of the security properties as efficiency is what impacts the experience of an end user most tangibly on a day to day basis. Thus, heuristic schemes are often deployed instead of schemes with provable security guarantees. Heuristic schemes are those that rely on intuitive notions of what constitutes privacy and can be quite fast, but it is difficult to analyze the exact privacy loss that results. A common deployed example of these are mix networks [23]. There are many cases of extracting patterns that break the privacy of heuristics in settings such as anonymous routing [47], encrypted communications [31], and electronic cash [63].

On the other hand, schemes that do receive rigorous academic attention have their own challenges. These works do add formalism, however emphasis tends to be on constructing schemes in new models and having asymptotic gains. While these focuses will result in undoubtedly better cryptographic protocols, they can often have poor concrete efficiency or are difficult to implement. Furthermore, there is a tendency for "folklore" solutions to arise, where implementation details are treated as trivial, but lack of deployment means that the viability of these approaches are not vetted.

This tension between practice and research is undesirable as it means that the deployed privacy-preserving systems end users have available to them are often not backed by the strong guarantees that cryptographic research is centered around. This can expose users to serious privacy loss as mentioned above. One approach to handle this issue is continued work in improving the concrete efficiencies of theoretical constructions so that they are viable in real world systems. However, a parallel concern is that even if an academic scheme is concretely efficient, there may be enough discrepancy in its model and the real world that implementers have challenges adapting the protocol to their applications. These are not simply engineering problems, but core structural questions that impact the security achieved. In this thesis, we focus on an approach we refer to as *practice-oriented privacy* as a means to alleviate this latter concern.

We take inspiration from the approach favored by practice-oriented provable-security (POPS) [1]. POPS focuses on making it easier to apply provable secure cryptographic constructions in practice by adding more details about concrete efficiencies and how to rely on the security of underlying primitives. In contrast, we focus more on looking at how to design systems with formal models of privacy that effectively map to real world use cases. We approach this from two directions. We analyze existing privacy-preserving systems to measure both their privacy guarantees as well as how they are used. We then look at solutions to these problems in the

literature and see if we can find non-trivial "gaps" in their models and explore ways to build an augmented system that more clearly applies to practice.

In particular, we focus on the settings of privacy-preserving payments and communication:

Design and applications of a blockchain analysis platform. Blockchains and cryptocurrencies have been a dramatic new development in the space of privacy-preserving payments. Different cryptocurrencies advertise varying amounts of anonymity as well as other functionalities; however, formal guarantees of these properties are uncommon. The public nature of the transaction graph on most blockchains offers an opportunity to systematically measure how these systems are being used. In the first part of this dissertation, we introduce BlockSci, a blockchain analysis platform. We show how the similar design of many blockchains allow for the construction of an analysis tool flexible enough to support multiple blockchains while still able to benefit from various optimizations. We then show how our tool can be used to perform analyses on the privacy and usage of these blockchains. In particular, we look at the privacy of the Dash cryptocurrency and provide an improved assessment of the velocity of cryptocurrencies. Our first analysis indicates that Dash's PrivateSend may still be vulnerable to clustering attacks that have been applied to other cryptocurrencies. Our second analysis indicates that a significant fraction of transactions on Bitcoin are "self-churn," transactions where the sender and receiver are the same party.

Reducing Bandwidth in Mixing Cryptocurrencies. The next part of the dissertation continues to take a look at the privacy-preserving payments setting, looking at Mixing Cryptocurrencies in particular. These are schemes which use cryptographic techniques to correlate the real input into a transaction with

many other "cover" inputs to hide who is actually performing the payment. Unfortunately, many of these schemes suffer from a practical limitation: the size of the transaction grows linearly with the size of the anonymity set. We propose a simple technique for efficiently sampling cover traffic from a finite (and public) set of known values, while deriving a compact description of the resulting transaction set. This technique, which is based on programmable hash functions, allows us to dramatically reduce transaction bandwidth when large cover sets are used. We refer to our construction as a recoverable sampling scheme. We present formal security definitions; prove our constructions secure; and show how these constructions can be integrated with various currencies and different cover sampling distributions.

Detecting Man-in-the-Middle via Network Latency and Reverse Turing Tests. The last part of this dissertation focuses on the setting of privacy-preserving communications. We investigate the problem of establishing secure communication channels in the challenging environment where participants do not share secrets or have access to trustworthy public-key distribution. In this setting, which includes many secure messaging and telephony protocols, participants cannot rule out the presence of a man-in-the-middle attacker without complex out-of-band checks. In practice, these checks rely on informal assumptions of the difficulty for an adversary to interfere with the check. In this work, we build on techniques originally proposed by Chaum [22] and Rivest and Shamir [81] to secure these communications using human authentication. Specifically, we develop and analyze key exchange and communication protocols that enable human participants to detect the presence of an attacker, by forcing the attacker to actively modify the contents of the (human-recognizable) communications that take place on the channel. We note the existing techniques from the literature do not specify what this detection process looks like and how

it ties into the system's security. We formalize and extend these techniques to a practical real-time setting, implementing our protocol on top of WebRTC, a real-time communication standard used in practice, providing a detailed security model for our proposal, and proving our protocols secure in this model.

We now delve into these problems in more detail.

1.1 Design and applications of a blockchain analysis platform

Public blockchains constitute an unprecedented research corpus of financial transactions. Bitcoin's blockchain alone is 260 GB as of December 2019.[1] This data holds the key to measuring the privacy of cryptocurrencies in practice, studying user behavior with regards to security and economics, or understanding the non-currency applications that use the blockchain as a database.

We present BlockSci, a software platform that enables the science of blockchains. It addresses three pain points of existing tools: poor performance, limited capabilities, and a cumbersome programming interface. Compared to the use of general-purpose graph databases, BlockSci is hundreds of times faster for sequential queries and substantially faster for all queries, including graph traversal queries. It comes bundled with analytic modules such as address clustering, exposes different blockchains through a common interface, collects "mempool" state and imports exchange rate data, and gives the programmer a choice of interfaces: a Jupyter notebook for intuitive exploration and C++ for performance-critical tasks. In contrast to commercial tools, BlockSci is not tailored to specific use cases such as criminal investigations or insights for cryptocurrency traders. Instead, by providing efficient and convenient programmatic access to the full blockchain data, it enables a wide range of reproducible, scientific

[1] All numbers in this paper are current as of December 2019, and analyses of the Bitcoin blockchain as of block height 610,695, unless stated otherwise.

analyses.

BlockSci's design starts with the observation that blockchains are append-only databases; further, the snapshots used for research are static. Thus, the ACID properties of transactional databases are unnecessary. This makes an in-memory analytical database the natural choice. On top of the obvious speed gains of memory, we apply a number of tricks such as converting hash pointers to actual pointers and deduplicating address data, which further greatly increase speed and decrease the size of the data. We plan to scale vertically as blockchains grow, and we expect that this will be straightforward for the foreseeable future, as commodity cloud instances currently offer up to a *hundred times* more memory than required for loading and analyzing Bitcoin's blockchain. Avoiding distributed processing is further motivated by the fact that blockchain data is graph-structured, and thus hard to partition effectively. In fact, we conjecture that the use of a traditional, distributed transactional database for blockchain analysis has infinite COST (Configuration that Outperforms a Single Thread) [62], in the sense that no level of parallelism can outperform an optimized single-threaded implementation.

BlockSci comes with batteries included. First, it is not limited to Bitcoin: a parsing step converts a variety of blockchains into a common, compact format. Currently supported blockchains include Bitcoin, Bitcoin Cash, Bitcoin SV, Litecoin, and Zcash. A multi-chain mode optimizes for user-friendly and memory-efficient analyses of forked blockchains together with their parent chain. Smart contract platforms such as Ethereum are outside our scope.

Second, BlockSci includes a library of useful analytic tools, such as identifying special transactions (e.g., CoinJoin) and linking addresses to each other based on well-known heuristics, including across forked chains. Third, BlockSci can record the time of transaction broadcasts on the peer-to-peer network and expose them through the same interface. Similarly, we make (historical and current) data on the exchange

rates between cryptocurrencies and fiat currencies readily available. These allow many types of analyses that wouldn't be possible with blockchain data alone.

The analyst begins exploring the blockchain through a Jupyter notebook interface, which initially exposes a `chain` object, representing the entire blockchain. Startup is instantaneous because transaction objects are not initially instantiated, but only when accessed. Iterating over blocks and transactions is straightforward, as illustrated by the following query, which computes the average fee paid by transactions in each block mined in December 2019:

```
fees = [mean(tx.fee for tx in block) for block in chain.range('Dec 2019')]
```

This interface is suitable for exploration, but for analyses requiring high performance, BlockSci also has a C++ interface. For many tasks, most of the code can be written in Python using a "fluent interface", an API design pattern that combines expressiveness and high performance.

We present various applications to illustrate the capabilities of BlockSci. In particular, we provide evidence that the cluster intersection attack reported recently [34] also works against Dash, a prominent privacy-focused altcoin with built-in mixing. Additionally, we provide improved estimates of the velocity of cryptocurrencies, i.e., the frequency with which coins change possession. This helps us understand their use as a store of value versus a medium of exchange.

1.1.1 Our Contributions

More concretely, in this thesis we focus on the following contributions:

Enabling the analysis of multiple blockchains. While Bitcoin and other blockchains provide a large quantity of public data, existing tools are either limited in scope, hard to use, or inefficient. In optimizing BlockSci for blockchain analysis, we found ways to take advantage of shared invariants among many

blockchains. This means that these different blockchains can be parsed into our system, benefit from the same optimizations, and have the same interface exposed to analysts. Thus, queries can be quickly and seamlessly applied to several blockchains. This allows for not only deeper analyses of the various blockchains in the ecosystem, but also more focused comparisons between these blockchains as well.

An analysis of Dash PrivateSend. One prominent privacy-focused cryptocurrency we analyze is Dash. It shares many similar design elements as Bitcoin, with some deviations in aspects like the PoW algorithm used. One significant difference is the distinction of certain nodes as Masternodes. These are full nodes which maintain a significant stake in the currency and participate in the consensus algorithm and facilitate special kinds of transactions. One of these transaction types is PrivateSend, Dash's anonymous payment functionality. Here, a series of Masternodes operate as "mixes," taking in inputs from multiple parties and creating a multiple input and multiple output transaction which obscures the linkage of each parties input to transaction output. Despite this, we were able to use BlockSci to successfully implement the cluster intersection attack which has been able to trace payments on other blockchains. Critically, we found that the attack has a high success probability on the default PrivateSend parameters at the time. This indicates PrivateSend is likely providing much less privacy than assumed.

Improved estimates of the velocity of cryptocurrencies. Crucial to understanding the privacy affording by blockchains and cryptocurrencies is understanding how they are being used. Many assumptions of the security and privacy of cryptocurrencies follow from the assumption that they are being used as currencies. For instance, a technique for achieving anonymous payments on the

platform may make assumptions about the regularity of transactions and the uniformity of different types of transactions. If these assumptions are wrong, then these techniques will stand out and be easy to detect. Thus, it is important to analyze the velocity of cryptocurrencies. If transactions where value moves between two different parties is rare, it will be difficult to hide such payments in the network. We use BlockSci to carry out this analysis by measuring the amount of "self-churn," or transactions where the sender and receiver are the same. We find that the velocity of cryptocurrencies like Bitcoin are much lower and more stable than naively computing the total transaction output per day would indicate.

1.2 Reducing Bandwidth in Mixing Cryptocurrencies

Cryptocurrencies such as Bitcoin suffer from well-known privacy limitations. These stem from the fact that each transaction on the currency's public ledger is explicitly linked to one or more preceding *transaction outputs* from which funds originate. A number of academic works [5, 50, 63, 67, 84] and for-profit companies [19, 28] have demonstrated that sensitive payment information can be extracted from the resulting public transaction graph.

Several recent currencies address this problem by directly incorporating cryptographic mixing into the consensus protocol. The underlying protocols, which include Zerocash [86], CryptoNote [85], Zerocoin [64] and RingCT [70] obscure the identity of the previous transaction output(s) being consumed—henceforth referred to in this paper as the *real* outputs—by hiding them within a larger set of cover outputs, which are other transactions on the ledger that are not being spent in this transaction. This practice hides the origin of funds by hiding the real outputs within a larger anonymity set [76]. In practice this technique is frequently realized using

non-interactive witness-indistinguishable (WI)[2] or zero knowledge (ZK) proofs, such as zkSNARKs [72]. Regardless of the exact technology employed for the proof system, transactions in these systems can be viewed as making the following statement:

> *This transaction references M "real" previous transaction outputs $(I_1, \ldots I_M)$ embedded within a public "transaction output list" $\mathcal{T}$ of previous outputs drawn from the ledger.*

While the protocols above use different techniques, for the purposes of this paper we will consider only two aspects: (1) the size of the list $\mathcal{T}$ for each transaction (which we denote by $|\mathcal{T}|$), and (2) the size of the resulting transaction as a function of $|\mathcal{T}|$. The former directly affects the privacy provided by the protocol: larger values of $|\mathcal{T}|$ may permit a larger anonymity set for each transaction. At the same time, limited space on the ledger can make larger transactions unworkable.

Protocols such as Zerocash and Zerocoin [64, 86] set $\mathcal{T}$ to be the set of of *all* previous outputs, using cryptographic accumulators and succinct proofs. This maximizes $|\mathcal{T}|$ and minimizes transaction size, though at the cost of using strong cryptographic assumptions. By contrast, "mixing" protocols such as CryptoNote and RingCT (used by Monero [67]) use a smaller cover set. In these protocols, the spender randomly samples a cover set for *each* transaction and transmits the description of $\mathcal{T}$ as part of the transaction data. The need to encode a new subset $\mathcal{T}$ within each transaction creates tension between the size of the anonymity set and the transaction size.

In this work, we focus exclusively on currencies that follow the mixing approach. Specifically, we address the following problem: using current approaches, the transaction bandwidth needed to describe $\mathcal{T}$ grows as $O(|\mathcal{T}|)$. While this is acceptable for small cover sets, it makes these protocols unworkable with larger amounts of cover traffic.[3] This is problematic, given that there is impetus within the development

[2] These WI proofs are often used as part of a larger primitive, such as a ring signature [82].

[3] In practice this set is generally encoded using $|\mathcal{T}|$ differentially-encoded transaction indices.

community to greatly increase the amount of cover traffic used in currencies such as Monero — to values as large as $|\mathcal{T}| = 100,000$ [13] — by incorporating asymptotically more efficient proof systems where the proof size is constant or logarithmic in $|\mathcal{T}|$ [14, 94, 100]. As this work is deployed, the description of $\mathcal{T}$ will increasingly dominate as the source of transaction bandwidth, and may become the effective limit on the size of the anonymity set.

1.2.1 Our Contributions

In this work we describe an alternative approach to sampling the transaction output list $\mathcal{T}$ such that the resulting description of $\mathcal{T}$ is *compact*. Our approach is relatively simple, although the details and security proofs require considerable attention. Rather than sampling $\mathcal{T}$ using the current approach, we propose to sample and encode this set using a programmable keyed hash function (see *e.g.,* [41, 42]) with a relatively short key K. We show that using this approach, the description of $\mathcal{T}$ can be structured so that it grows with the number of real transaction outputs (M), rather than with the number of total elements in the list (N). For large cover sets, our approach should represent a significant improvement in space efficiency over the naïve approach currently used by real currency systems [15, 67].

Of course, this approach cannot be achieved using any hash function. In this work we show how to construct such a function, as part of a protocol that we call a *recoverable sampling scheme*. We provide security definitions and security proofs for our constructions in the random oracle model.

We discuss our approach in two settings. First, we consider an approach that samples the cover traffic *uniformly* from the set of all previous transaction outputs, which closely matches the approach used by CryptoNote (as implemented in the ByteCoin currency) [15, 85]. We then move to a more complex (and realistic) setting

Accumulator-based currencies work around this issue, since the set $\mathcal{T}$ can be represented by a single value indicating the current transaction's position in the ledger.

where $\mathcal{T}$ is sampled according to a specific and *non-uniform* distribution. Finally, we discuss concrete scheme parameters and give bandwidth cost estimates for integrating our approach into various deployed currencies, including ByteCoin [15] and Monero [67]. As a final matter, we describe applications of our technique to other application areas, including anonymous communications and Client-Server Puzzles [49].

1.3 Detecting Man-in-the-Middle via Network Latency and Reverse Turing Tests

The global SARS-CoV-2 pandemic has re-emphasized the importance of secure and highly-available video communications. In April 2020 researchers announced that the encryption keys for Zoom calls were generated by centralized servers, and discovered that certain Zoom calls between US participants were routed via servers in Beijing [59]. Concerns over the security of video calls produced an uproar in the United States and Europe [108, 109], and resulted in a rapid multi-company effort to roll out end-to-end encrypted (E2EE) video calling in products such as Zoom and Microsoft Teams [32, 107].

Deploying end-to-end encryption in video conferencing system requires designers to grapple with several technical challenges. Chief among these is the need to securely agree on shared encryption keys, in an environment where *all centralized infrastructure may be under adversarial control.* This threat model is challenging due to the fact that customers may not be able to trust centralized identity management, and can potentially be vulnerable to MitM attacks by centralized key servers. The designers of Zoom's End-to-End encryption protocol [6] acknowledge this, and propose various transparency, SSO and "key fingerprint" mechanisms as defense against such misbehavior.

Despite these efforts, defenses against MitM attacks in current-generation E2EE video calling largely devolve to provider trust, or (in systems such as Zoom's) to an

awkward procedure that involves comparing a security code verbally or through an out-of-band channel. Such mechanisms, seem highly unscalable. In this work we consider an alternative approach to solving this problem.

The challenge of preventing MitM attacks. The notion of public-key key agreement was first proposed by Diffie and Hellman more than 40 years ago [24], and forms the basis of a vast number of secure communication protocols. In the intervening decades, much subsequent work in this area has focused on *authenticated* key agreement (AKE) [2, 53, 54], in which one or both parties cryptographically verify their identity using a private key or shared secret. It is well known that some form of authentication is critical to prevent active attacks on key agreement protocols.

While standard AKE protocols are ideal for establishing secure channels, such protocols are not viable in all real-world settings. In many common applications the parties cannot exchange a shared secret or public key in a secure fashion. For example, Zoom and Teams users rely on centralized infrastructure to authenticate other parties, set up the communications, and to forward all sent packets to the correct party. Because users do not currently have public key certificates provided by an external party, they are completely reliant on the provider behaving honestly for the key exchange to not be corrupted.

As an alternative to trusting the channel, an alternative approach is to first perform an *unauthenticated* key exchange to establish a shared secret, and only later authenticate the channel to rule out the possibility that an active attack may have occurred during the key exchange. This approach is used in Zoom end-to-end encryption, as well as other messaging and video systems including OTR messaging [11], Signal [91], and ZRTP telephony [16].

This later authentication may involve comparing key fingerprints via an out-of-band channel, or performing an in-band comparison of a Short Authentication String

(SAS) using the human voice [96]. Because the authentication may occur after the key agreement has completed, we refer to such protocols as *a posteriori* authenticated key agreement protocols. Here, the objective is not preventing an active attack outright, but rather using the guaranteed detection of an attack occurring to dissuade an adversary.

While *a posteriori* authenticated key agreement has proven popular in practical applications, the existing techniques have several limitations. In some cases, the parties may not possess a secure secondary channel to compare key fingerprints. Some research has demonstrated that Short Authentication String (SAS) comparison can be foiled using speech synthesis techniques [89]. In practice, of course, the most common threat to these systems is simple laziness: research demonstrates that only a small percentage of OTR and ZRTP avail themselves of these verification mechanisms in the first place [87, 97]. This motivates the need for alternative techniques.

Ideally, these authentication techniques would have attack detection built-in without the need for explicit human intervention. Additionally, their security should rely on a challenge that would be difficult for an adversary to solve without breaking some strong assumption.

Employing human authentication. An important realization for building systems with otherwise weak security guarantees was that in many scenarios the adversary is an automated program with strict limitations on its ability to imitate human behaviors. This suggests that *Hard AI* problems, problems that are currently consider infeasible for AI but are tractable for humans, can be leveraged to harden systems against these automated adversaries. Naor introduced this idea in 1996 [69] and proposed several potential problems that could be used. Perhaps most famously, CAPTCHAs formalize this notion and introduce the idea of using distorted images as a way to stop bots from interacting with online services [98]. Recently there has been interest in building cryptography on top of CAPTCHAs and related problems [18, 55]. In these schemes,

the security of the system is formally tied to the hardness of some AI problem. We detail related work in this space that is tangential to our contributions in Section 5.6.

These Hard AI problems have been informally applied to *a posteriori* authenticated key agreement in the form of *forced latency* protocols introduced by Rivest and Shamir's Interlock [81] and further developed by Wilcox-O'Hearn [101] and Chaum [22, 88]. These *forced latency* protocols rely on the idea that messages generated by an adversary would result in "semantic irregularities" that are detectable by the honest parties. A communicating party sends a commitment to a message, waits some interval of time, and then reveals the message. If an adversary wishes to modify this message, they must wait to obtain the message before sending over the commitment to the receiving party. After waiting the established interval of time, it then sends over the message to the receiver. Thus, in these protocols the presence of an active attacker significantly increases the latency of the communication. By including key information in the commitment, communicating parties can use a forced latency protocol to detect whether an attack on the key agreement occurred.

While these forced latency protocols are very interesting as *a posteriori* authenticated key agreement protocols, they lack a great deal of formalism. The essential notion of "semantic irregularities" is not well defined nor have the protocols been clearly formalized or proven. Moreover, these protocols were proposed in an era when online communication was primarily conducted through textual means, and even in this limited setting the results were not evaluated. Modern Internet communications include full-motion video and audio calling, which can greatly enhance the potential of these ideas, provided that appropriate protocols can be developed for this new setting.

Strengthening latency protocols. In this work we propose a new class of protocol for *a posteriori* key agreement and encryption that is designed to facilitate secure audio/video communications. Our protocol achieves strong security guarantees in a setting where parties do not possess reliable public key distribution, under the

(non-cryptographic) assumption that two communicating parties can distinguish the behavior of an impersonating adversary over the course of a communication session. Unlike previous techniques, our approach does not depend on an optional SAS comparison ceremony, nor does it require an out-of-band channel.

1.3.1 Our Contributions

More concretely, our contributions are as follows:

Modeling human authentication. We place the work of Chaum and Rivest and Shamir on firmer ground, by providing a model that formalizes the "semantic irregularity" into a well-defined problem of recognizing whether you are having a conversation with a specific human. We then present a complete framework for secure channel establishment called *human authenticated secure communications* (HASC) that models a secure communication scheme based on human authentication.

The LATENT protocol. We describe a general framework for using human authentication to detect man-in-the-middle attacks on secure communications. The resulting protocol approach, called LATENT, uses committing encryption to ensure that attackers must either (1) emulate the behavior of a specific human being, or else (2) engage in attacks that produce measurable effects on the protocol, specifically an increase in end-to-end latency. We formally define the resulting protocol and prove its security in the HASC model.

LATENT-WebRTC. While the previous contributions help to place forced latency protocols onto firmer theoretical ground, they leave an important question: do these protocols work in practice? To explore this, we implement a prototype variant of the LATENT protocol for secure video, modifying the WebRTC standard to incorporate our protocol ideas. We then measure the effectiveness of

our protocol at detecting interception by a sophisticated attacker who can inject artificial network delay in an effort to fool our detection. We provide empirical measures of our approach and evaluate the system end-to-end.

1.3.2 Limitations

We want to be clear about the limitations of our work. Our work does not imply that our LATENT protocol is future proof. The landscape of what is considered a Hard AI problem is constantly evolving, and it is unlikely that many of today's problems will continue to be hard down the line. We believe our work is valuable for the following reasons:

1. We believe that providing a way to formally connect the security of an authentication protocol to a Hard AI problem is a worthwhile exercise. While other systems such as ZRTP or Interlock attempt to make use of human authentication techniques, the lack of formalism means it is not clear if solving the related Hard AI problem is a necessary and sufficient condition for breaking the system.

2. Even if our approach does not remain secure in the long term, designing a system for today's environment lays the foundation for continuing to integrate Hard AI techniques in authentication. CAPTCHAs are a prominent example of the benefits of a formal connection between cryptography and Hard AI problems, and while the original CAPTCHA designs are no longer secure, there continues to be work [30] in the search for novel ways to build CAPTCHAs.

3. While our setting does not capture all possible network scenarios, we believe attempting to model around common network conditions is still instructive. Indeed, our assumptions still lead to many nontrivial design decisions, and we believe these serve as an important foundation for designing more robust systems.

1.4 Bibliographic Notes

The result on BlockSci and its application is based on contributions to joint work [50, 51] with Harry A. Kalodner, Malte Möser, Kevin Lee, Steven Goldfeder, Martin Plattner, and Arvind Narayanan that appeared in USENIX 2020. The result on reducing bandwidth in mixing cryptocurrencies is based on joint work [21] with Matthew Green that appeared in IEEE European Symposium on Security and Privacy Workshop on Security and Privacy on the Blockchain 2018. The result on detecting Man-in-the-Middle attacks using network latency and reverse Turing tests is based on joint work with Matthew Green that is currently in submission.

1.5 Outline of the Thesis

In Chapter 2 we begin by recalling some cryptographic preliminaries and giving an overview of blockchains. In Chapter 3, we present our results on building and applying a blockchain analysis platform. In Chapter 4, we present our result in reducing bandwidth in mixing cryptocurrencies. Finally, we present our results on how to detect Man-in-the-Middle attacks using network latency and reverse Turing tests in Chapter 5.

Chapter 2

Background

2.1 Cryptographic Preliminaries

In what follows, a PPT adversary $\mathcal{A}$ is a probabilistic interactive Turing Machine that runs in polynomial time in the security parameter λ. We will drop the security parameter λ from the notation when it is implicit.

2.1.1 Diffie-Hellman Contributory Key Exchange

Let $\mathbb{G}$ be a group of prime order q. Let g be a generator of $\mathbb{G}$. A Diffie-Hellman Key Exchange between two parties A, B consists of the following steps:

1. Each party generates a Diffie-Hellman key pair as follows: $a \xleftarrow{\$} \mathbb{Z}_q$ (resp. $b \xleftarrow{\$} \mathbb{Z}_q$). (a, g^a) is the key pair for party A (resp. (b, g^b) is the key pair for party B).
2. The two parties exchange the public key share g^a (resp. g^b) with the other party.
3. The shared key is computed as g^{ab}

The contributory property of Diffie-Hellman Key Exchange is described informally as the requirement that neither party can force the derived key to be a particular value. In the two party case, this is extended to claim an adversary cannot conduct a

key exchange with two parties independently and have them derive the same key[1]. We capture this property in the following definition.

Definition 2.1. *We say that the Diffie-Hellman Key Exchange is contributory if for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that*

$$Pr\left[g^{ab'} = g^{a'b} \middle| \begin{array}{l} (a, g^a) \leftarrow A; (b, g^b) \leftarrow B \\ a', b' \leftarrow \mathcal{A}(1^\lambda, g^a, g^b) \end{array}\right] \leqslant \nu(1^\lambda)$$

2.1.2 Pseudo-Random Function

A *pseudo-random function* is an algorithm PRF. This algorithm implements a deterministic function $y = \mathsf{PRF}(k, x)$, taking as input a key $k \in \mathcal{K}_{\mathsf{PRF}}$ and some bit string x, and returning a string $z \in \{0,1\}^\mu$.

Definition 2.2. *Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.*

1. *The challenger samples $k \xleftarrow{\$} \mathcal{K}_{PRF}$ uniformly random.*
2. *The adversary may query arbitrary values x_i to the challenger. The challenger replies to each query with $y_i = \mathsf{PRF}(k, x_i)$. Here, i is an index, ranging from $1 \leq i \leq q$ for some $q \in \mathbb{N}$. Queries can be made adaptively.*
3. *Eventually, the adversary outputs value x and a special symbol $\perp$. The challenger sets $y_0 = \mathsf{PRF}(k, x)$ and samples $y_1 \xleftarrow{\$} \{0,1\}^\mu$ uniformly random. Then it tosses a coin $b \xleftarrow{\$} \{0,1\}$, and returns z_b to the adversary.*
4. *Finally, the adversary outputs a guess $b' \in \{0,1\}$.*

A pseudo-random function is said to be secure if for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that

[1] We ignore the trivial case where the adversary sets both keys to g^0

$$|Pr[b' = b] - 1/2| \leqslant \nu(1^\lambda)$$

2.1.3 Authenticated Encryption with Associated Data

We follow the definition of Rogaway [83] where an authenticated encryption scheme with associated data is an tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ that are defined as follows:

- $\mathsf{KeyGen}(1^\lambda, K) \to k, n$: Takes in as input 1^λ where λ is the security parameter and a long term key K. Then it samples $n \xleftarrow{\$} \{0,1\}^\ell$, where ℓ is the length of nonces for this AEAD scheme. It returns $k := \mathsf{KDF}(K, n)$ and n.
- $\mathsf{Enc}(k, M, D) \to C$: Takes as input a key k, a plaintext M, additional data to authenticate D and returns a ciphertext C.
- $\mathsf{Dec}(k, C, D) \to M \in \{\perp\} \cup \{0,1\}^*$: Takes in a key k, a ciphertext C, and additional data to authenticate D and if the ciphertext was validly generated under k it returns the plaintext. Otherwise it returns $\perp$.

For correctness we required that $\mathsf{Dec}(k, \mathsf{Enc}(k, M, D), D) = M$. We informally restate the security properties of an AEAD scheme. An AEAD scheme has *privacy* if an $\mathcal{A}$'s advantage in distinguishing between encryptions of different plaintexts $\mathbf{Adv}^{PRIV}$ is bounded by a negligible function. An AEAD scheme has *non-malleabilty* if for a ciphertext $C \leftarrow \mathsf{Enc}(k, M, D)$, $\mathcal{A}$'s advantage in producing a C', D' such that $D' \neq D$ and $\mathsf{Dec}(k, C', D') \neq \perp$, $\mathbf{Adv}^{AUTH}$, is bounded by a negligible function.

We note that in practice an AEAD scheme satisfying these properties is often considered to be *committing encryption*, where the ciphertext is bound to a message and the key is considered an opening for the commitment. However, there have been some recent works [20, 27] that found that some AEAD schemes do not meet strong notions of committing encryption. While a general AEAD scheme is sufficient for the setting in this work, caution must be taken to not conflate these two primitives.

2.1.4 Collision-Resistant Hash Functions

A *collision-resistant hash function* is a function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$.

Definition 2.3. *Consider the following security experiment $CR_{\mathcal{H}}^{\mathcal{A}}$ with an adversary $\mathcal{A}$.*

1. *The adversary $\mathcal{A}$ outputs a pair of messages X, X'.*
2. *If $\mathcal{H}(X) = \mathcal{H}(X')$ and $X \neq X'$, output 1, otherwise output 0.*

A collision-resistant hash function is said to be secure if for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that

$$Pr[CR_{\mathcal{H}}^{\mathcal{A}} \Rightarrow 1] \leqslant \nu(1^\lambda)$$

2.1.5 Ring Signatures

A standard ring signature scheme comprises three (possibly) probabilistic algorithms:[2]

- $\mathsf{KeyGen}(1^\lambda)$. On input a security parameter λ, outputs a keypair pk, sk.
- $\mathsf{RSign}((pk_1, \ldots, pk_n), j, sk, m)$: Given a set of public keys $(pk_1, \ldots, pk_n)$, a message m and the index j and secret key of the signer sk, outputs a ring signature σ.
- $\mathsf{RVerify}((pk_1, \ldots, pk_n), m, \sigma)$: On input a set of public key $(pk_1, \ldots, pk_n)$, a signature σ and a message m, outputs 1 if the signature is valid and 0 otherwise.

There are many ring signature variants, and each offers different features. Informally, all ring signatures are expected to satisfy at least the following properties:

[2]Some ring signatures also require a global Setup algorithm that generates a common reference string (CRS). We omit this here.

- **Correctness:** Any honestly generated ring signature should be considered valid by any verifier.
- **Unforgeability:** Adversaries should have negligible probability of forging a ring signature. Here forgery is defined as producing a ring signature for a message m and ring R without the signer being a member of R. Unforgeability must hold even when the adversary can adaptively choose messages and groups to obtain ring signatures on.
- **Anonymity:** All adversaries (who may be other ring members) should have at a most negligible advantage in identifying the true signer.

2.2 Blockchains

Blockchains are decentralized append-only databases, sometimes also referred to as a decentralized ledger. A blockchain consists of a series of records, known as blocks, where each block contains a cryptographic hash of the previous block, timestamp, and transaction data. This securely links the blocks together as altering a prior block would require altering all subsequent blocks as well. The system is made up of a decentralized peer-to-peer network of nodes, where each node has a local copy of the blockchain. Updates to the blockchain are performed through a consensus protocol, where nodes come together to validate and add new blocks. In practice, the state of the blockchain is what the majority of the nodes agree upon. The two prominent use-cases of blockchains are cryptocurrencies and smart contracts.

A cryptocurrency is a blockchain where blocks are generated through a mining process. This mining is part of the consensus algorithm and dictates who gets to generate the next block. The two main ways this is determined is Proof-of-Work, where a node demonstrates it has completed some computational task, and Proof-of-Stake, where a node is selected based on its current stake in the system. Miners, the nodes

who are able to generate the next block, obtain currency from both a reward for generating the block and transaction fees, a fee paid by the transactions included in the block. Users can obtain cryptocurrency from exchanges, where other currencies are exchanged into a cryptocurrency. They can then pay other users on the network by creating transactions, where a payment is not considered complete until it is included in a block. This enables for a robust payment system without the need for a central intermediary.

Smart contracts on the other hand, are a form of distributed application. In these systems, users can create executable scripts to be included in blocks. These scripts can then be executed by the blockchain network. For example, one could generate an escrow protocol that sends value from one party to another if a specified condition is met. While smart contract platforms, such as Ethereum, are an important part of the blockchain ecosystem, they are out of scope for our work.

2.2.1 Bitcoin-like Blockchains

Bitcoin [68] can be considered the parent of modern blockchain technologies. Most current blockchains take at least some, if not a majority, of their design choices from Bitcoin. Bitcoin operates with a Proof-of-Work consensus protocol, with a new block being added roughly every 10 minutes. Users are identified by their bitcoin address, a pseudonym that is used to specify the sender and receiver of a transaction. Transactions are made up of a series of inputs and outputs, where the input and output must have the same total. Transaction outputs (TXOs) assign value to a specific address. Transaction inputs are all references to previous unspent transaction outputs (UTXOs), where in order to spend an input, the spender must show they control the address of the associated output. Blockchains that follow this design, in particular the "each input spends one output" UTXO paradigm, are considered Bitcoin-like. Examples of Bitcoin-like blockchains are Litecoin, Namecoin, and Dash.

These blockchains generally support basic scripts that determine how outputs can be redeemed. A common script is the "pay-to-public-key-hash" (P2PKH), where to spend an output, the redeemer must provide both a public key that corresponds to the included hash as well as a digital signature using the corresponding private key. Other blockchains add further scripts to extend how the cryptocurrency can be used or to add privacy.

2.2.2 Monero

Monero is a privacy-centered cryptocurrency that is based on the CryptoNote protocol [85]. While it shares many design similarities to Bitcoin, it makes a fundamental change that puts it in a separate class of protocols. This change is its use of ring signatures in the redemption of outputs. Rather than the "one input spends one output" paradigm of Bitcoin, these ring signatures allow a spender to provide a set of possible outputs that are being spent with a proof that they control one of the outputs in that set. This allows the sender to hide which outputs they control, providing privacy. Consequently, there is no longer a well defined UTXO set as it is unclear which outputs have actually been spent and which are being used as a cover set. However, Monero includes a mechanism to still prevent double spending of any outputs. Monero additionally supports stealth addresses, cryptographically derived one-time addresses that hide whether multiple payments were made to a single address. Additionally, Monero supports confidential transactions where the value of the inputs and outputs in a transaction are hidden as well. As a result, all outputs look the same to an outsider, even if they hold different amounts of value. This greatly simplifies the process of making a cover set when creating a transaction.

Chapter 3

Design and applications of a blockchain analysis platform

3.1 Design and architecture

Figure 3-1 shows an overview of BlockSci's architecture. There are two routes for importing data into BlockSci (Section 3.1.1). Through either route, the data is converted by the parser into the BlockSci Data (Section 3.1.2), which can be incrementally updated as new blocks come in. The analysis library loads this data as an in-memory database, which the user can either query directly (in C++) or through a Jupyter notebook interface (Section 3.1.4).

A recurring theme in this section is that since BlockSci is a domain-specific database, we are able to make assumptions about the schema and the workload that allow us to achieve large performance gains and an expressive interface. Both this broad lesson and some of our specific optimizations may be applicable to other domains.

3.1.1 Recording and importing data

Design decision: which blockchains should BlockSci support? There are hundreds of blockchains, some of which differ from Bitcoin in minor ways and others drastically. As we aim to provide a common interface for the analysis of all supported blockchains, supporting too few blockchains (e.g., just Bitcoin) limits usefulness, but

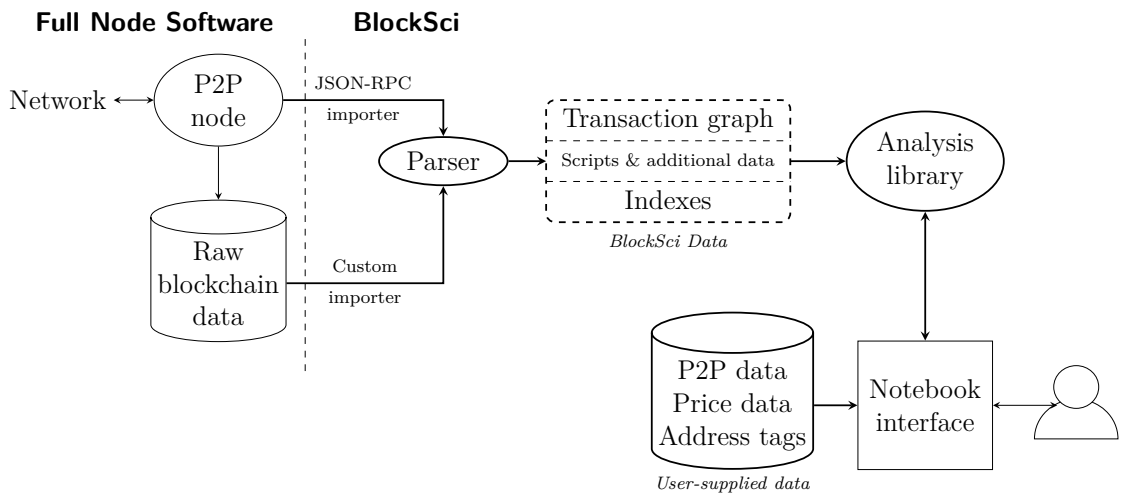

Figure 3-1. Overview of BlockSci's architecture.

supporting too many different blockchains would complicate the interface and make optimizations harder.

Recall that the Bitcoin blockchain consists primarily of a directed acyclic graph of transactions. The edges connecting transactions have attributes, i.e., addresses or scripts, attached to them. Transactions are grouped into blocks which are arranged in a linear chain, with a small amount of metadata per block. BlockSci supports blockchains that follow this basic structure. For example, Litecoin makes no changes to the data structure, and is thus fully supported. Cryptocurrencies that introduce changes to the script operations may be supported only partially, but the user can parse unknown scripts with a few lines of code. Zcash is also supported, at least to the extent that Zcash blockchain analysis is even possible: it introduces a complex script that includes zero-knowledge proofs, but these aspects are parceled away in a special type of address that is not publicly legible by design.

An example of an unsupported blockchain is Monero, as it doesn't follow the "each input spends one output" paradigm. Its transaction graph contains additional edges, the mixins. Supporting it would require changes to the data layout as well as the programmer interface. Similarly, Ethereum departs from the transaction-graph model,

and further, its script is vastly different from and more complex than that of Bitcoin.

In our analyses we have worked with six blockchains: Bitcoin, Bitcoin Cash, Litecoin, Namecoin, Dash, and Zcash. Many other cryptocurrencies make no changes to the blockchain format, and so should be supported with no changes to BlockSci.

Multi-chain mode. By default, BlockSci operates on a single blockchain. We also provide a multi-chain mode in which several forked chains (e.g., Bitcoin $\prec$ Bitcoin Cash $\prec$ Bitcoin SV) can be combined in an optimized, memory-efficient multi-chain configuration. In this mode, data common to forked chains (such as pre-fork transactions) need to be loaded into memory only once. Address data is deduplicated across forks, allowing for novel cross-chain analyses.

Importer. For cryptocurrencies with small blockchains where import performance is not a concern, we use the JSON-RPC interface. The advantage of this approach is versatility, as many cryptocurrencies aim to conform to a standard JSON-RPC schema regardless of the on-disk data structures and serialization format. For larger blockchains (currently only Bitcoin and its forks are large enough for import performance to be a concern), we use our own high-performance importer that directly reads the raw data on disk.

Mempool recorder. BlockSci can optionally record mempool data, that is, timestamps of transactions that are broadcast to the P2P network and are waiting to be included in the blockchain. The waiting time of included transactions provides valuable data for economic analyses and isn't recorded in the blockchain itself. When users choose to collect these timestamps, they are accessible through the same interface as all other blockchain data.

3.1.2 BlockSci Data

Key challenge: finding a data layout that gives a good trade-off between memory efficiency and performance.

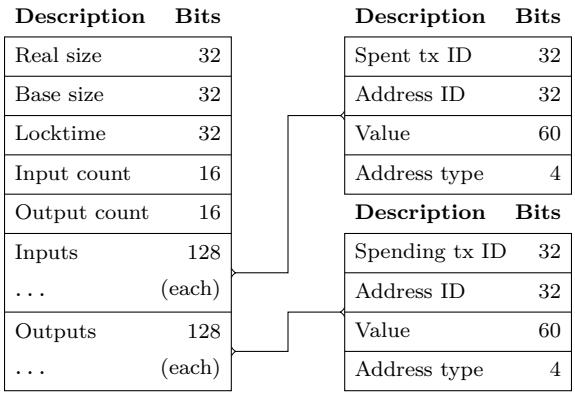

Figure 3-2. Transaction structure

Based on our experience with empirical blockchain analysis over several years, we divide the available data into three categories and combine it in a hybrid scheme that provides us with a reasonable trade-off between efficient use of memory and speed of access:

1. The core transaction graph is required for most analyses and always loaded in-memory. It is stored in a row-based format.

2. Scripts and additional data is required for only a subset of analyses. It is stored in a hybrid (partially column-based, partially row-based) format and is loaded on-demand.

3. Indexes to look up individual transactions or addresses by hash are stored in a separate database on disk.

We make further optimizations to improve performance, including using fixed-size encodings for data fields where possible, optimizing the memory layout for locality of reference, linking outputs to inputs for efficient traversal, and sharing identical data across chains in multi-chain mode.

Transaction graph. The core transaction graph is stored in a single sequential table of transactions, with entries having the structure shown in Figure 3-2. Note that entries have variable lengths, due to the variable number of inputs and outputs (there

is a separate array of offsets for indexing, due to the variable entry lengths). Normally this would necessitate entries to be allocated in the heap, rather than contiguously, which would have worse memory consumption and worse locality of reference.

However, because of the append-only property of the blockchain, there are only two types of modifications that are made to the transactions table: appending entries (due to new transactions) and length-preserving edits to existing entries (when existing UTXOs are consumed by new transactions). This allows us to create a table that is stored as flat file on disk that grows linearly as new blocks are created. To load the file for analysis, it is mapped into memory. The on-disk representation continues to grow (and be modified in place), but the analysis library provides a static view.

Layout and locality. The main advantage of the transaction graph layout is spatial locality of reference. Analyses that iterate over transactions block-by-block exhibit strong locality and benefit from caching. Such analyses will remain feasible even on machines with insufficient memory to load the entire transaction graph, because disk access will be sequential.

The layout stores both inputs and outputs as part of a transaction, resulting in a small amount of duplication (a space cost of about 19 %), but resulting in a significant speedup for sequential iteration compared to a normalized layout. Variants of the layout are possible depending on the types of iteration for which we wish to optimize performance.

Additional data. Beyond the core transaction graph, BlockSci provides access to additional data that are necessary for some types of analyses. These include script data, transaction hashes and version numbers, input sequence numbers, input-output linkages, and raw data contained in coinbase transactions. Keeping this data separate reduces memory usage in exchange for a small reduction in speed of access for analyses that require this data (e.g., 10 % slower for a typical query that iterates over transaction metadata).

Scripts. BlockSci categorizes scripts into 5 generic types, each of which contains scripts of one or more address formats: script-hash (for script-hash and witness-script-hash scripts), pubkey (for raw pubkey, pubkey-hash, individual pubkeys in a multisig script, and witness-pubkey-hash scripts), multisig, null data, and unknown witness scripts. All other scripts are categorized as nonstandard. Internally, script data of different address formats is deduplicated: for example, a public key used in both a pubkey-hash and a witness-pubkey-hash script is stored only once. For nonstandard scripts, BlockSci stores the entire script data which can be parsed with only a few lines of code by the analyst.

Indexes. Transaction hashes and addresses are stored in flat files and can easily be looked up by transaction/address ID. The reverse mapping from hash to ID, however, is stored in separate indexes in RocksDB databases (the address index is also used by the parser). Accessing these indexes is almost never performance critical in scientific analysis—in fact, many analyses don't require the indexes at all. Besides the ability to look up transactions and addresses by hash, we also provide a lookup for all outputs associated with specific addresses.

Multi-chain mode. To support forked blockchains, we make three modifications to the layout described above. First, forked chains often share a large common history with their parent chain. We load these identical blocks only once, and the analysis library provides the abstraction of a full chain for each fork. Second, the fixed-size encoding does not permit storing data of multiple chains. For example, UTXOs at fork height can be spent in both the parent and the forked chains, but the fixed-length field can only hold a single index for the spending transaction (cf. Figure 3-2). Each fork thus needs a separate flat file that contains the spending transactions' IDs for outputs created before the fork. Third, the index that maps addresses to outputs requires an additional chain identifier to distinguish between outputs on different chains.

3.1.3 Address Linking

Address linking (or clustering) is a key step in many analytic tasks including understanding trends over time and evaluating privacy. Recall that cryptocurrency users can trivially generate new addresses, and most wallets take advantage of this ability. Nevertheless, addresses controlled by the same user or entity may be linked to each other, albeit imperfectly, through various heuristics.

Two common types of heuristics include (1) inputs spent in the same transaction are controlled by the same entity, and (2) identifying a change address based on client software or user behavior (e.g., [63]). As the multi-input heuristic does not apply to CoinJoin transactions, we add an exception for those transactions, which we identify using the algorithm described by Goldfeder at al. [34]. Change address identification is challenging due to the variety of existing client software. BlockSci comes with several—as of this writing, ten—change address heuristics that can be used individually or in combination with each other, allowing the analyst to choose or create a heuristic best suited for their analysis task.

These heuristics create links (edges) in a graph of addresses. By iterating over all transactions and applying the union-find algorithm on the contained addresses we generate clusters of addresses. This set of clusters is the output of address linking. We use the union-find implementation by Jakob [45]. Clustering takes only a few minutes, allowing the analyst to recompute and compare clusters with different heuristics.

In multi-chain mode, BlockSci can enhance the clustering of a target chain using information from forked chains. Addresses that exist on multiple chains may be used differently on them, e.g., combined with a different set of input addresses. Cross-chain address clustering uses these additional links to enhance the clustering of the target chain.

Figure 3-3 shows the distribution of cluster sizes for Bitcoin using the multi-input

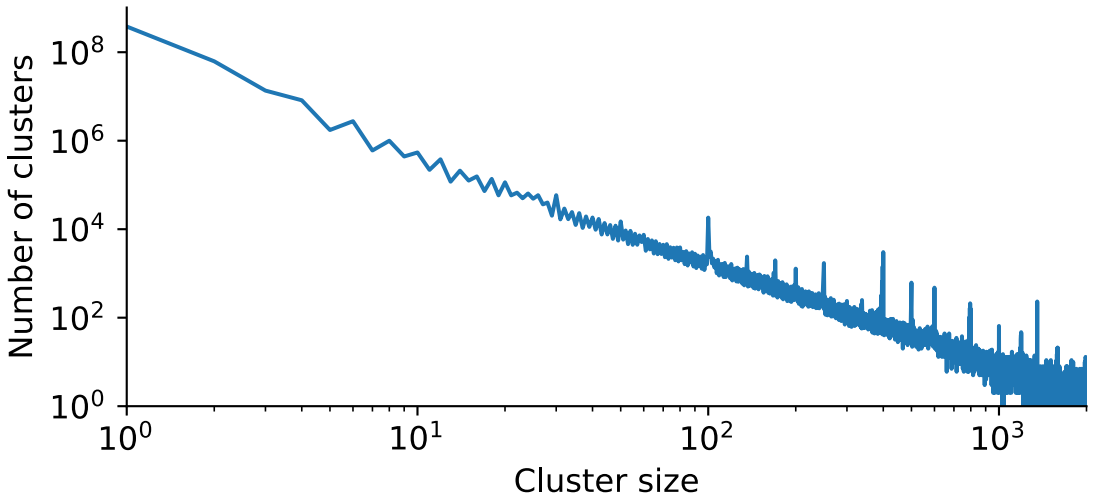

Figure 3-3. Distribution of sizes of address clusters in Bitcoin after applying address-linking heuristics. Sizes 1–2,000 are shown here but there are many clusters that are much larger.

heuristic only. There are about 474 million clusters in total, of which about 380 million are single addresses, and about 93 million have between 2 and 20,000 addresses. There are 809 clusters with over 20,000 addresses, including one supercluster with over 17 million addresses.

Address linking is inherently imperfect, and ground truth is difficult to obtain on a large scale, since it requires interacting with service providers. We do not attempt to be comprehensive, resulting in false negatives (i.e., missed edges, resulting in more clusters than truly exist). More perniciously, most of the heuristics are also subject to false positives (i.e., spurious edges), which can lead to "cluster collapse". In particular, it is likely that the supercluster above is a result of such a collapse.

3.1.4 Programmer interface

Key challenge: combining speed and expressiveness. BlockSci aims to come close to the speed of C++ while providing expressiveness and convenience of a high-level language, namely Python, for as many analysis tasks as possible.

Python interface. Jupyter notebook is a popular Python interface for data science. It allows packaging together code, visualization, and documentation, enabling easy sharing and reproducibility of scientific findings. We expose the C++ BlockSci library to Python through the pybind11 interface [46]. While we intend Jupyter notebook to be the main interface to BlockSci, it is straightforward to utilize the analysis library directly from standalone C++ or Python programs and derive most of the benefits of BlockSci.

Python is not a language known for performance; unsurprisingly, we find that it is significantly slower to run queries through the Python interface. Nevertheless, our goal is to allow the programmer to spend most of their time interacting with the Jupyter notebook, while simultaneously ensuring that the bottleneck parts of queries execute as C++ code. We illustrate this through an example.

Suppose our goal is to find transactions with anomalously high transaction fees — say 0.1 bitcoins (10^7 satoshis), worth about 720 US dollars as of December 2019. The slowest way to do this would be to write the entire query in Python:

```
[tx for block in chain for tx in block if sum(txin.value for txin in
    tx.inputs) - sum(txout.value for txout in tx.outputs) > 10**7]
```

This way does not result in acceptable performance. A first step to improve both performance and conciseness is to have a built-in function to compute the fee:

```
[tx for block in chain for tx in block if tx.fee > 10**7]
```

Although `tx.fee` calls a C++ function, we model it as a property in the Python interface. Most helper functions are modeled as properties, unless they are expected to take significant time to compute, or take arguments. `tx.fee` is just one of many helpers exposed by the Python library that execute as C++. We've found that most of the analyses in Section 3.2 benefit from a small number of helper functions.

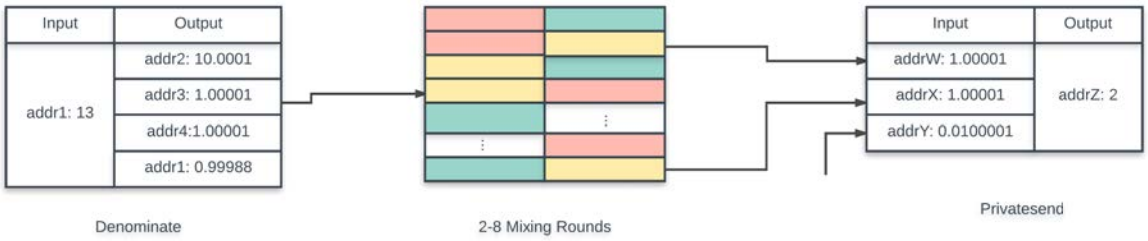

Figure 3-4. Overview of Dash privacy. First, in the Denominate step, a coin is broken down into valid denominations and the remainder is returned to the original address. Here, `addr2`, `addr3`, and `addr4` are the new denominated coins and the leftover 0.99988 Dash is sent back to `addr1`. Then for each denominated coin, there will be 2–8 rounds of mixing. When a user wishes to make a PrivateSend, the wallet will use these mixed coins as inputs. The input amount must be a multiple of the smallest denomination. Additionally another mixed input will be included as a fee. Here, the first two inputs provide the value for the output. The third input is for the fee. This value will generally be 0.0100001 Dash, but if coins of that denomination are not available, the wallet selects a mixed coin of the smallest denomination it possesses.

3.2 Applications

We now present two analyses that highlight BlockSci's effectiveness at supporting blockchain analyses. The first relates to privacy and confidentiality, the second to the economics of cryptocurrencies. Together these provide a clearer picture behind how blockchains are used in practice.

3.2.1 Cluster intersection attack on Dash

Goldfeder et al. recently showed the effectiveness of the cluster intersection attack against Bitcoin mixing [34]. The attack seeks to link mixed coins to the cluster of wallet addresses that originally held the coins before mixing. The intuition behind the attack is that outputs mixed in different transactions are often spent together. Thus, when these coins are spent together, we trace each one back to a (potentially large) set of possible address clusters and examine the intersection of these sets. This will likely result in a unique cluster. We conclude that the mixed outputs are linked to the wallet represented by this cluster.

Figure 3-5 PrivateSend wallet simulation.
Input: desired amount to spend in a PrivateSend
Output: a set of unspent outputs to add up to this value

```
1: procedure SelectPSInputs(send_amount)
2:     T ← set of transactions that have at least one
           output that is unspent and owned by us
3:     T ← Sort T by (denomination, transaction hash)
4:     selected ← {}
5:     for each t ∈ T do:
6:         for each output ∈ t.outputs do:
7:             if value(selected) + value(output)
8:                  > send_amount then
9:                 break
10:            end if
11:            selected.insert(output)
12:            if value(selected) == send_amount then
13:                return selected
14:            end if
15:        end for
16:    end for
17:    return "Insufficient Funds"
18: end procedure
```

This is a significant weakness of mixing as an anonymity technique. In this section we provide evidence that Dash, a cryptocurrency designed with mixing in mind, is susceptible to this attack.

Overview of Dash. Dash is one of three popular privacy-focused altcoins (alternative cryptocurrencies), along with Monero and Zcash. It is the largest of the three by market capitalization as of August 2017 — over USD 2 billion. It is supported by a handful of vendors and a few alternative payment processors [90]. Dash is a fork of Bitcoin with a few key changes. It has a shorter block time (from 10 to 2.5 minutes) and uses the X11 hashing algorithm. It also has a two-tiered network, where nodes controlling 1,000 Dash or more have the option of becoming "Masternodes" — full nodes that participate in the consensus algorithm, facilitate special types of transactions, and get a cut of the mining reward for their service. One of these special types of transactions is PrivateSend.

Dash's PrivateSend uses CoinJoin-style mixing, whereas Monero uses mixing based on ring signatures and Zcash provides cryptographic untraceability, which is a stronger (and provable) anonymity property. Mixing is not mandatory in Dash, but it is integrated into the default wallet and therefore easy to use. When a user chooses to start mixing, all her coins (up to a configurable limit with a large default value) are mixed with several rounds of mixing. The number of rounds is also configurable, but the default is 2. These mixed coins are then available for PrivateSend transactions.

Mix transactions in Dash use power-of-10 denominations. Therefore coins are broken up into these standard sizes before mixing is initiated. The mix transactions themselves each have three participants, each of whom contributes between 5 and 9 coins to be mixed. Finally, the PrivateSend transactions spend a set of mixed power-of-10 denominated outputs. Each of these three types of transactions has a distinct signature that is readily detectable on the Dash blockchain. In particular, the denominations are $1.00001 * 10^k$ instead of exactly 10^k, and thus the values are highly

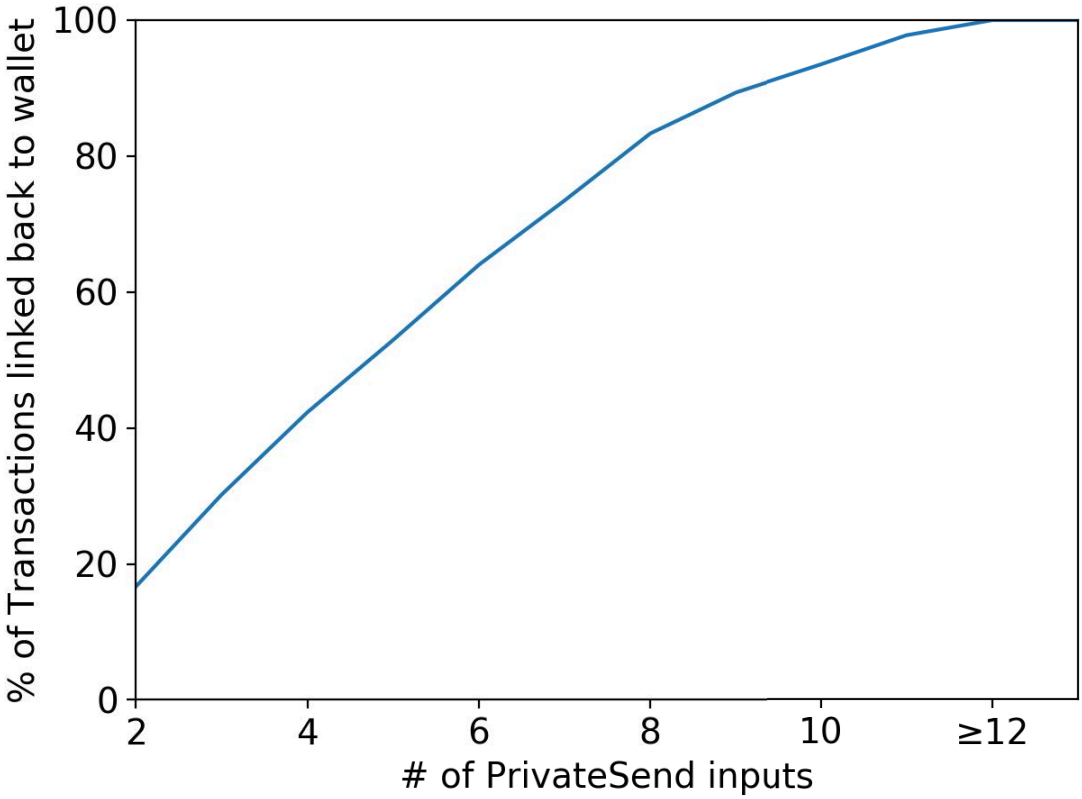

Figure 3-6. Success rate of the cluster intersection attack on simulated Dash PrivateSend transactions as a function of the number of inputs.

unlikely to occur by chance. See Figure 3-4.

Dash and cluster intersection. Two features of the PrivateSend implementation combine to make Dash especially vulnerable to the cluster intersection attack. First, change addresses are not allowed for these transactions. This means that PrivateSend spenders must produce "exact change", which requires combining a large number of coins. Second, the denominations being powers of 10 (as opposed to, say, powers of 2) further increases the number of inputs in a typical transaction. For example, to pay 85 Dash, the sender must combine at least 8+5=13 inputs to avoid losing money. Figure 3-7 shows the distribution of the number of inputs in PrivateSend transactions. Most such transactions have 3 or more inputs; the mean is 40.1 and the median is 12.

Due to the large number of inputs, no auxiliary information is necessary to carry

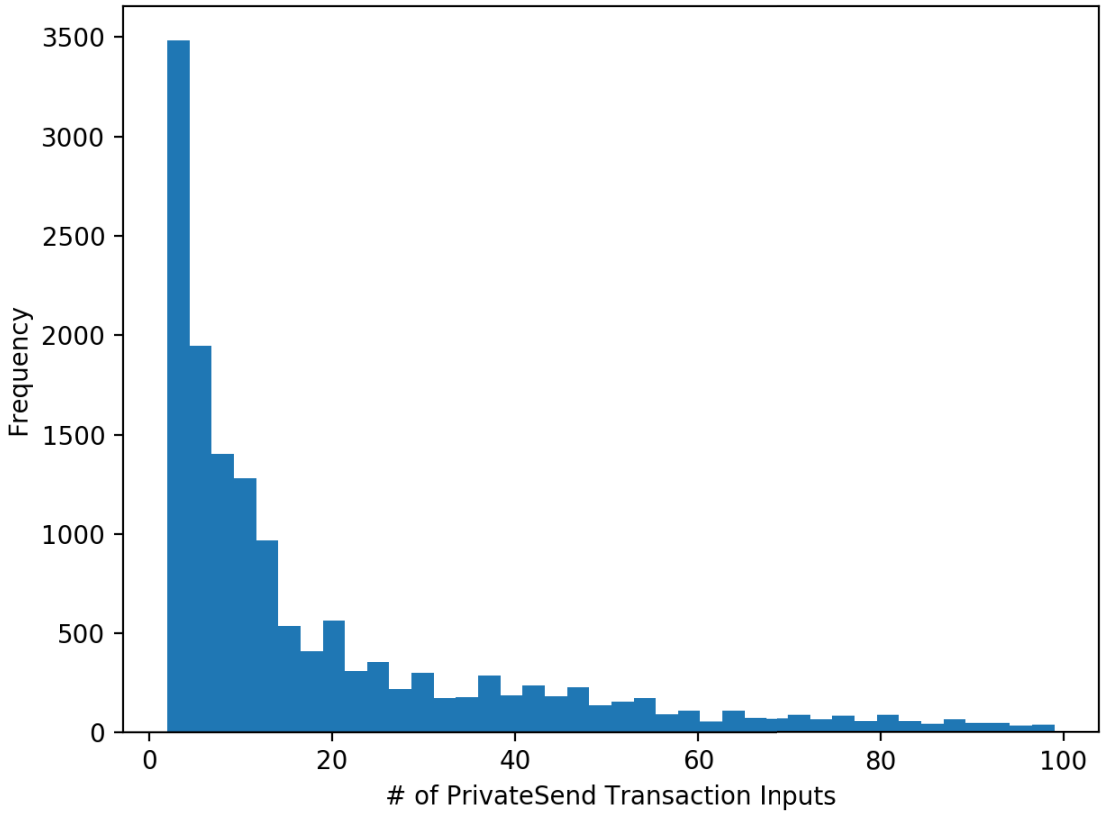

Figure 3-7. Distribution of the number of inputs of Dash PrivateSend transactions

out the cluster intersection attack on Dash. The adversary — anyone observing the public blockchain — can infer that all inputs to a PrivateSend must trace back to the same wallet cluster. Thus, in the above example of a payment of 85 dash, the adversary knows that all 13 sets of clusters must have an element in common. The chance that there is more than one such cluster gets smaller and smaller as the number of clusters increases.

Of course, auxiliary information can make this attack more powerful. Beyond the risks posed by tracking cookies in [34], the Masternodes learn the input-output linkage for the mixing rounds that they facilitate. The privileged status of Masternodes in the Dash p2p network raises other potential privacy vulnerabilities [25], but that is not our focus.

Experimental setup. To perform this attack, we used shapeshift.io (an online service for conversion between cryptocurrencies) to convert Bitcoin into Dash, which we withdrew into a single address. We used the default Dash wallet to mix 0.55 Dash using the default parameters, namely 2 rounds of mixing. We obtained 55 separate mixed outputs, each 0.01 Dash.

Next, we re-implemented the PrivateSend algorithm from the Dash wallet code on top of BlockSci. Given a desired spend amount, the algorithm selects a set of mixed inputs from the wallet that sum to this amount. It is shown in Figure 3-5. This allowed us to simulate our own PrivateSend transactions instead of actually making them. The latter would have required paying a transaction fee for each data point; generating the data shown below would have required spending several hundred USD worth of Dash in transaction fees, and holding several tens of thousands of USD worth of Dash.

For each of the simulated PrivateSends, we ran the cluster intersection attack. We consider the attack successful if it results in a unique cluster of addresses, namely the single address that we started from.

Results. Figure 3-6 shows the success rate of the cluster intersection attack, showing a sharp increase in accuracy as the number of inputs increases. For transactions with 12 or more inputs (coincidentally, the median number of inputs of PrivateSend transactions on the blockchain), the attack is always accurate.

In the above experimental setup, we started from a single pre-mixing address holding Dash. In reality, users may obtain Dash in multiple installments and hold these coins in their wallet in a manner that is not easily linkable to each other. Relying on this is unwise for privacy, as it is a form of security through obscurity; nevertheless, it is a factor that will significantly hurt the accuracy of the attack in practice. Evaluating the attack on existing PrivateSend transactions is challenging due to the lack of ground truth, and is a topic for future work.

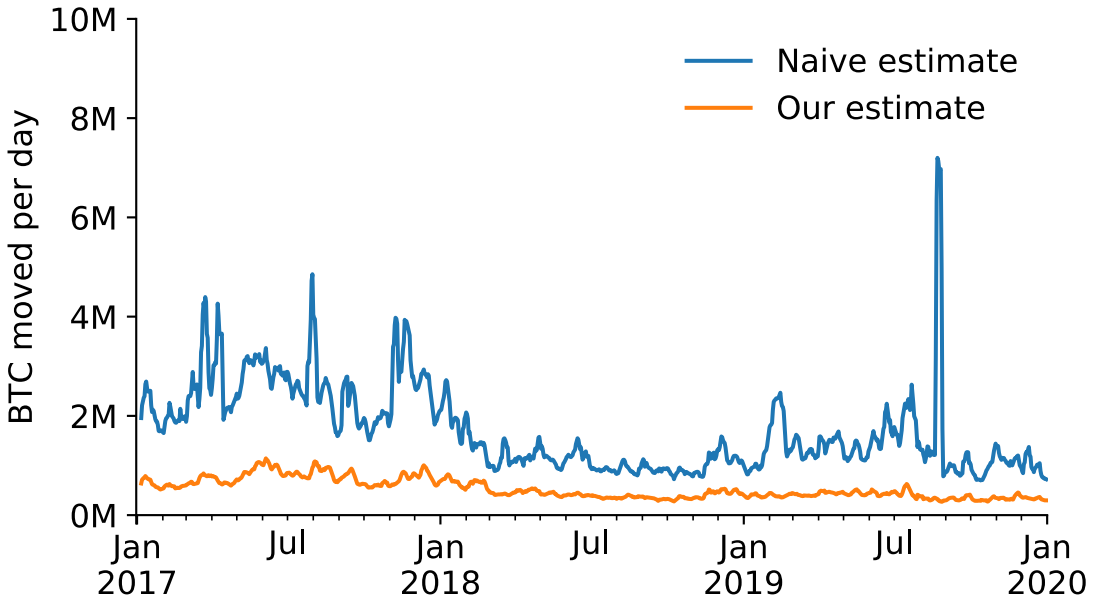

Figure 3-8. Two estimates of the velocity of bitcoins.

3.2.2 Improved estimates of the velocity of cryptocurrencies

The velocity of money is the frequency with which one unit of currency is used for purchases in a unit of time. It can provide an insight into the extent to which money is used as a medium of exchange versus a store of value.

In most cases it is not possible to infer the purpose behind a cryptocurrency transaction from the blockchain. However, an alternative definition of the velocity of money is the frequency with which one unit of currency changes possession in any manner (whether or not for purchases of goods and services) in a unit of time. Blockchain analysis may enable estimating the velocity of cryptocurrencies under this definition.

Even under this simplified definition, it is challenging to estimate the velocity of cryptocurrencies. A naive method would be to compute the total value of transaction outputs in a unit of time and divide it by the total value of the money supply during that period. However, multiple addresses may be controlled by the same entity, and

therefore not all transaction outputs represent changes in possession. Meiklejohn et al. call this "self-churn" [63], a term that we adopt. The impact of self-churn is visually obvious in the graph of total transaction outputs (Figure 3-8). We would not expect spikes such as those in early 2017 if the graph reflected actual money demand, which would be much more stable over time.

To minimize the effect of self-churn, we adopt two heuristics. First, we eliminate outputs controlled by an address that can be linked to one of the inputs' addresses (through address clustering), ignoring "superclusters" to minimize false positives. This reduces change outputs and transactions that are detectable as an entity "shuffling their money around". We also eliminate outputs that are spent within less than k blocks (we use $k = 4$). Manual examination suggests that such transactions are highly likely to represent self-churn, such as "peeling chains" where a large output is broken down into a series of smaller outputs in a sequence of transactions.

The orange line in Figure 3-8 shows the daily transaction volume on the Bitcoin blockchain after applying the above two heuristics. With this estimate, the velocity of Bitcoin works out to 1.2 per month averaged over the period January 2017–June 2018, compared to 3.9 with the naive metric, and 0.7 over the period July 2018–December 2019, compared to 2.2 with the naive metric. Our revised estimate is not only much lower but also much more stable over time.

Starting in 2018 the naive estimate drops closer to our improved estimate. We suppose that this is partially due to scarcity in block space (and a corresponding rise in transaction fees), which encourages intermediaries to batch multiple payments into a single transaction, thereby eliminating some of the self-churn that is evident in the naive estimate earlier. Spikes in the graph, like the one in mid 2019, may represent large intermediaries (e.g., exchanges) moving large amounts of bitcoin to addresses with updated access control structures.

We note several caveats. First, this still likely fails to exclude some transfers of

value between addresses controlled by the same entity. Without ground truth, it is hard to be certain how good the estimate is. Second, it doesn't count transfers of possession that don't touch the blockchain. When exchanges, online wallets, and other intermediaries hold money on behalf of users, payments and transfers of "bitcoins" might happen even though no actual bitcoins changed hands (as only account balances in an internal database need to be updated). Nevertheless, we believe that the metric can be a useful proxy for understanding the use of cryptocurrencies, and possibly for comparing between cryptocurrencies.

3.2.3 Other applications of BlockSci

Besides our own use, BlockSci has seen a variety of use in both academic and industry settings. We are currently aware of at least 9 peer-reviewed articles, 6 preprints, and 2 software projects that use BlockSci for blockchain analysis (a full list is available online[1]).

The dual topics of privacy and forensics are common among these papers. These include information leaks from payments and purchases through intermediaries [34], the use of intermediaries to convert between cryptocurrencies [105], as well as the identification of entities and the analysis of their behavior in the transaction graph [12, 29, 43, 48, 106]. Many of these results are of interest to law enforcement and regulators, and we have helped regulators use BlockSci for their own investigations. Two other themes are issues surrounding the security and scalability of cryptocurrencies [73, 74, 93], as well as economic analyses of cryptocurrencies [4].

BlockSci has also been used as the foundation for specialized blockchain analysis tools. Boshmaf, Al Jawaheri, and Al Sabah [12] have built a tagging system on top of BlockSci, and the GraphSense blockchain analytics platform uses BlockSci's parser and altcoin support to generate an address graph out of the transaction graph [39].

[1] https://citp.github.io/BlockSci/studies/

3.3 Conclusion

There is a high level of interest in blockchain analysis among developers, researchers, and students, leading to an unmet need for effective analysis tools. While general-purpose in-memory graph databases exist, a tool customized to blockchain data can take advantage of its append-only nature as well as provide integrated high-performance routines for common tasks such as address linking.

BlockSci has already been widely used as a research and educational tool. We hope it will continue to be broadly useful, and plan to keep maintaining it as open-source software.

Chapter 4

Reducing Bandwidth in Mixing Cryptocurrencies

4.1 Intuition

At a high level, our approach is straightforward: rather than sampling and transmitting the full set $\mathcal{T}$ with each transaction, we define a specialized hash function $H_K(\cdot)$ with some useful properties. To construct a transaction and transaction list $\mathcal{T}$, the spender first constructs and transmits a key K within the transaction, along with an integer N. The nature of the function is such that, given this key, any party can now efficiently compute $\mathcal{T}$ as $(H_K(0), H_K(1), \ldots, H_K(N-1))$.

The challenge is to construct the function H such that the resulting set $\mathcal{T}$ will embed up to M "real" transactions chosen by the spender, without revealing *which* transactions are the real ones, and which are cover traffic.[1] To do this, we must solve several problems. First, we require a function H with a compact key K that can be programmed to incorporate the real transactions at random points. Second, we must ensure that the remaining "cover" transactions are all sampled from the appropriate distribution (which may not be uniform). Finally, we need to prove that this technique is as secure as the naïve sampling approach; *i.e.,* that it does not leak new information

[1] In practice, this guarantee depends largely on the distribution of the real transactions. We define an approach that guarantees our technique is "no worse" than the naïve sampling approach.

to an attacker who wishes to identify the real transactions.

As a warm up, we begin by describing a simple technique for sampling the cover set uniformly from the list of all transaction outputs on a ledger consisting of ℓ such outputs. For purposes of exposition, we will consider the simple case where there is only one real transaction ($M = 1$), and we will assume that all parties know the list of previous transactions.

As an ingredient we require an underlying keyed hash function $f : \{0,1\}^\kappa \times \mathbb{Z}_N \to \mathbb{Z}_\ell$, which we model as a random oracle.[2] We now define a simple programmable hash function $H : \{0,1\}^\kappa \times \mathbb{Z}_\ell \times \mathbb{Z}_N \to \mathbb{Z}_\ell$ as follows:

$$H_{k,C}(i) = f_k(i) + C \ mod \ \ell$$

Notice that a spender, who wishes to embed some real transaction index $I \in [0, \ell)$ in $\mathcal{T}$, can easily program this function as follows: sample a random $k \in \{0,1\}^\kappa$ and a random index $j \in [0, N)$. Now compute a value C such that $C \equiv I - f_k(j) \ mod \ \ell$. This implicitly defines $H_k(j) \equiv I \ mod \ \ell$ and sets each remaining $H(i), i \neq j$ to be uniform in $\mathbb{Z}_\ell$.

Of course, this simple scheme works only for cases where there is a single real transaction. In practice, real protocols may require the spender to embed *several* real transactions into $\mathcal{T}$. In our main construction, we generalize the above construction by replacing the single value C with a polynomial $P(\cdot)$ evaluated over a field F_p for some large prime p. This approach allows us to ensure that *for any* subset of M distinct indices j, $f_k(j) + P(j) \equiv I_j \ mod \ \ell$. It remains only to prove that the resulting scheme is "as secure" as the naïve sampling approach. We provide a security definition for this claim, and offer a proof in the random oracle model.

Of course, this proposal is not complete. Given a solution such as the one above, we must still adapt the details so that they work within a real cryptocurrency. In

[2]Such functions can easily be constructed from any standard cryptographic hash function.

later sections of the paper we discuss altering the function H so that the cover traffic is sampled from a more realistic distribution. Additionally, we address the problem that our approach may produce *duplicate* transaction outputs within the transaction list $\mathcal{T}$, and discuss how to remove these. We provide a detailed discussion of these aspects of the problem in §4.5.

4.1.1 Outline of this work

The remainder of this work proceeds as follows. In the next two sections we provide definitions for our schemes. In §4.4 we provide a construction that assumes a *uniform* sampling distribution for cover transactions. In §4.5 we discuss sampling for alternative distributions. In §4.6 we show how our constructions can be integrated with specific cryptocurrency mixing protocols, and give concrete estimates of transaction size. In §4.7 we describe some different potential applications for these techniques outside of cryptocurrency privacy. Finally, §4.8 discusses related work.

4.2 Preliminaries

4.2.1 Notation

We will define $\nu(\cdot)$ to be a negligible function. Let $A \overset{c}{\approx} B$ indicate that the distributions A, B are computationally indistinguishable. We will use $\mathcal{D}$ to refer to a specific distribution for sampling transaction outputs.

4.2.2 Transaction sets

While thus far we have referred to transaction *sets*, in our main constructions we will relax the requirement that each element is unique, and we will add an implicit ordering. Henceforth we will consider $\mathcal{T}$ to be an ordered *multiset*, or merely a list of transactions.

4.2.3 Transaction ledger

We assume the existence of an append-only public ledger of transaction outputs $\mathbf{L} = (T_0, T_1, \ldots, T_{\ell-1})$ that is available to all parties in the system. New transaction outputs are appended to $\mathbf{L}$ after they have been verified by the network. By $\mathbf{L}_\ell$ we denote the first ℓ transactions on $\mathbf{L}$.

We will assume that each transaction output $T_i \in \mathbf{L}_\ell$ can be referenced uniquely by its index $I \in \{0, \ldots, \ell - 1\}$; henceforth we will use these indices exclusively to represent transaction outputs on the ledger.

4.2.4 Keyed hash functions with integer domain and range

Let m, n be positive integers. We define $f_{m,n} : \{0,1\}^\kappa \times \mathbb{Z}_m \to \mathbb{Z}_n$ as a keyed function that, on input a key $k \in \{0,1\}^\kappa$ and an integer $a \in \{0, \ldots, m-1\}$, outputs an integer $b \in \{0, \ldots, n-1\}$. We will assume that each pair (m, n) uniquely defines the function family, and that these functions can be re-constructed efficiently by any party.[3] In our security proofs we will model this function as a random oracle.

4.3 Definitions

The purpose of this work is to define an approach to sampling the transaction list that produces a compact description. We now define this scheme.

Definition 4.1. *A recoverable sampling scheme (RSS) is parameterized by a sampling distribution $\mathcal{D}$. It consists of two possibly probabilistic algorithms* $(\mathsf{Sample}, \mathsf{Recover})$ *with the following definition:*

$\mathsf{Sample}_{\mathcal{D}}(1^\lambda, \ell, \mathcal{I}, N) \to (\mathcal{T}, \mathsf{W})$. *On input a security parameter λ, a ledger size ℓ, a set of M legitimate transaction indices $\mathcal{I} = \{I_0, \ldots, I_{M-1}\} \in \mathbb{Z}_\ell^M$, and a desired*

[3] Note that functions of this form can be constructed efficiently from any standard underlying hash function using a variety of techniques.

number of total transactions $N > M$, this algorithm outputs a tuple (ordered multiset) $\mathcal{T}$ containing N (possibly non-unique) elements, and a compact description W.

$\mathsf{Recover}_{\mathcal{D}}(\mathsf{W}, \ell) \to \mathcal{T}$. *On input a compact description* W *and the ledger size* ℓ*, outputs the transaction multiset* $\mathcal{T}$ *or the distinguished failure symbol* $\perp$.

An RSS must satisfy three properties, which we refer to as *correctness*, *compactness*, and *security*. We define these below.

Correctness. Let $(\mathcal{T}, \mathsf{W}) \leftarrow \mathsf{Sample}_{\mathcal{D}}(1^{\lambda}, \ell, \mathcal{I}, N)$. For an RSS to be *correct*, several requirements must be met for all valid $(\lambda, \ell, \mathcal{I}, N)$. First, it must hold that $\mathcal{T}$ contains N elements and $\mathcal{I} \subset \mathcal{T}$. Finally, the following equality must be satisfied:

$$\mathsf{Recover}_{\mathcal{D}}(\mathsf{W}, \ell) = \mathcal{T}$$

Compactness. For an RSS to be useful, it must hold that given a fixed M the size of the compact description W should grow sublinearly as N increases. We note that the compactness requirement rules out a class of trivial schemes, including any scheme that simply outputs $\mathsf{W} = \mathcal{T}$.

Security. Intuitively, security for a recoverable sampling scheme requires that the scheme reveals "no more" information to an attacker than would be revealed by an ideal implementation that simply samples $N - M$ cover transactions and combines these with $\mathcal{I}$ in a random ordering. To define this form of security, we must first describe two experiments.

Real experiment. Let $(\mathcal{A}, \lambda, \ell, \mathcal{I}, N, \mathcal{D})$ be the input to the experiment. Compute $(\mathcal{T}, \mathsf{W}) \leftarrow \mathsf{Sample}_{\mathcal{D}}(1^{\lambda}, \ell, \mathcal{I}, N)$ and run $\mathcal{A}(\ell, |\mathcal{I}|, N, \mathsf{W})$. The output of the experiment is $\mathcal{A}$'s output.

Ideal experiment. Let $(\mathcal{B}, \lambda, \ell, \mathcal{I}, N, \mathcal{D})$ be the input to the experiment. Construct a multiset $\mathcal{C}$ consisting of $N - |\mathcal{I}|$ (possibly duplicate) transaction indices (sampled according to $\mathcal{D}$) from the set $\mathbb{Z}_\ell$; and shuffle these values randomly with the values in $\mathcal{I}$ to obtain the ordered list $\mathcal{T}$. Next run $\mathcal{B}(\ell, |\mathcal{I}|, N, \mathcal{T})$. The output of the experiment is $\mathcal{B}$'s output.

We are now prepared to define security for an RSS.

Definition 4.2 (Security for RSS)**.** *An RSS* $\Pi = (\mathsf{Sample}, \mathsf{Recover})$ *is secure if for every* p.p.t. *adversary* $\mathcal{A}$*, sufficiently large* λ*, and all valid* $(\ell, \mathcal{I}, N, \mathcal{D})$*, there exists a p.p.t.* $\mathcal{B}$ *such that the following holds:*

$$\mathbf{Real}(\mathcal{A}, \lambda, \ell, \mathcal{I}, N, \mathcal{D}) \stackrel{c}{\approx} \mathbf{Ideal}(\mathcal{B}, \lambda, \ell, \mathcal{I}, N, \mathcal{D})$$

Discussion. We note that this is a purely comparative definition. That is, our definition does *not* imply that the "ideal" sampling scheme is itself secure for any distribution or set of input transactions. Indeed, for many values of $\mathcal{I}$ the real transactions may be easily distinguishable from the cover transactions! Addressing that problem is not the purpose of this work: instead, our goal merely to show that the RSS scheme performs "no worse" than the ideal approach, while offering an improvement in bandwidth efficiency.

As an additional note, our ideal experiment implements sampling with replacement, resulting in the potential for duplicates in the transaction list $\mathcal{T}$. This is slightly different than some implemented schemes that remove duplicates. We make this relaxation order to simplify the presentation of the rest of this paper, although in later sections we address the problem of removing duplicates from this set. Finally, to simplify our definitions, we do not explicitly include auxiliary inputs to the parties. This must be done for sequential composition of the primitive. Our constructions achieve this notion as well.

$\mathsf{Sample}_{\mathcal{U}}(1^\lambda, \ell, \mathcal{I} = \{I_0, \ldots, I_{M-1}\}, N)$

1 : Choose a prime p such that $1/(p/\ell) \leq \nu(\lambda)$.
2 : Construct the keyed hash function $f : \{0,1\}^\kappa \times \mathbb{Z}_N \to \mathbb{Z}_p$.
3 : Sample a random key $k \in \{0,1\}^\lambda$.
4 : Sample random $\{j_0, ..., j_{M-1}\} \subset \{0, \ldots, N-1\}$.
5 : For $i = 0$ to $M-1$: Choose y_{j_i} such that $f_k(j_i) - y_{j_i} \equiv I_i \ mod\ \ell$.
6 : Compute $a_0, ..., a_{M-1}$ that define an order-$(M-1)$ polynomial $P(\cdot)$ over $\mathbb{F}_p$ s.t. $\forall i \in [0, M)$, $P(j_i) = y_{j_i}$.
7 : For $i = 0$ to $N-1$: compute $T_i = f_k(i) - P(i) \ mod\ \ell$.
8 : **return** $\mathcal{T} = \{T_0, \ldots, T_{N-1}\}$ and $W = (p, k, P, \ell, N)$.

$\mathsf{Recover}_{\mathcal{U}}(\mathsf{W}, \ell)$

1 : Construct the keyed hash function $f : \{0,1\}^\kappa \times \mathbb{Z}_N \to \mathbb{Z}_p$.
2 : For $i = 0$ to $N-1$: compute $T_i = f_k(i) - P(i) \ mod\ \ell$.
3 : **return** $\mathcal{T} = \{T_0, \ldots, T_{N-1}\}$.

Figure 4-1. The RSS uniform sampling and recovery algorithms

4.4 A Uniform Sampling Technique

We begin by describing a technique for sampling the cover set *uniformly* from the set of all transactions on a ledger $\mathbf{L}_\ell$. This technique mirrors the approach of certain protocols such as CryptoNote [85] as implemented in the ByteCoin currency [15]. The algorithms we describe below allow us to embed M real input transactions into a transaction multiset $\mathcal{T}$ of size N, where $0 < M < N$. We will assume that both spender and verifier have access to the ledger $\mathbf{L}_\ell$.

The sampling algorithm. Let $\mathbf{L}_\ell$ be the ledger. Let $I_0, ...I_{M-1} \in \mathbb{Z}_\ell$ represent the *indices* (into $\mathbf{L}$) of the legitimate input transactions. Let N represent the desired size of $\mathcal{T}$ and $M > 1$ represent the number of legitimate input transactions (where $M < N$). We will make use of a field $\mathbb{F}_p$ of prime order $p \gg \ell$ (such that $\frac{1}{p/\ell}$ is negligible). To sample this set, the spender performs the steps in Figure 4-1.

The recovery algorithm. The above algorithm produces a set $\mathcal{T}$ that can be used to construct the transaction. However, the full description of $\mathcal{T}$ need not be included in the transaction. Instead, the spender may include the tuple $W = (k, P, \ell, N)$. This

can be used by the verifier to recover $\mathcal{T}$ as described in Figure 4-1.

Efficiency. Our scheme requires sampling a random key, which can be done with marginal computational effort. In practice we can use an efficient algorithm such as AES to perform the function of f. Constructing the polynomial also requires minor computational effort. Using Lagrange polynomials it requires about $(n+1)(2n+1)$ multiplications in $\mathbb{F}_p$ to compute the encoding.

Correctness and compactness. Assuming a fixed M, the size of the compact description W clearly grows at most logarithmically as N increases.[4] It is easy to see that the algorithms above are *correct*, in the sense that $I_0, ..., I_{M-1}$ is contained within $\mathcal{T}$. Specifically, there exists $\{j_0, ..., j_{M-1}\} \subset \{0, \ldots, N-1\}$ such that for each j_i:

$$\begin{aligned} T_{j_i} &\equiv f_{N,\ell}(k, j_i) - P(j_i) \quad mod\ \ell. \\ &\equiv f_{N,\ell}(k, j_i) - y_{j_i} \quad mod\ \ell \\ &\equiv f_{N,\ell}(k, j_i) - f_{N,\ell}(k, j_i) + I_i \quad mod\ \ell \\ &\equiv I_i \quad mod\ \ell. \end{aligned}$$

4.4.1 Security

We now prove that the above scheme is a secure RSS in the sense of Definition 4.2.

Theorem 4.1. *The protocol* $\Pi = (\mathsf{Sample}, \mathsf{Recover})$ *described above is a secure RSS if the function* $f : \{0,1\}^\kappa \times \mathbb{Z}_N \to \mathbb{Z}_p$ *is modeled as a random oracle.*

Proof. To succeed in our proof, we must show that for every p.p.t. $\mathcal{A}$ there exists a p.p.t. adversary $\mathcal{B}$ such that for all p.p.t. distinguishers $\mathcal{Z}$ we have that $|\mathbf{Pr}[\mathcal{Z}(\mathbf{Real}(\mathcal{A}, \lambda, \ell, \mathcal{I}, N, \mathcal{U})) = 1] - \mathbf{Pr}[\mathcal{Z}(\mathbf{Ideal}(\mathcal{B}, \lambda, \ell, \mathcal{I}, N, \mathcal{U})) = 1]| \leq \nu(\lambda)$ over all valid $\ell, \mathcal{I}, N$. Let $\mathcal{A}$ be the adversary that interacts in the **Real** experiment. Given $\mathcal{A}$ we show how to construct $\mathcal{B}$, which satisfies the above requirement.

[4] And this is only because W contains a representation of N in order to simplify the description of the algorithm.

$\mathcal{B}$ conducts the **Ideal** experiment, and runs $\mathcal{A}$ (and answers its random oracle queries) as follows. When $\mathcal{B}$ receives $(\ell, |\mathcal{I}|, N, \mathcal{T})$ from the **Ideal** challenger, it parses $\mathcal{T} = (I_0, \ldots, I_{N-1}) \in \mathbb{F}_p^N$. It then selects p as in the real protocol, samples a random key $k \in \{0,1\}^\kappa$ and a random set of coefficients $(a_0, ..., a_{M-1}) \in \mathbb{Z}_p^M$ that define the polynomial $P(\cdot)$. For $i = 0$ to $N-1$, it programs the random oracle such that:[5]

$$f_k(i) \equiv P(i) + I_i \; mod \; \ell$$

Finally, it sets $\mathsf{W} = (p, k, P = (a_0, ..., a_{M-1}), \ell, N)$ and sends $(\ell, |\mathcal{I}|, N, \mathsf{W})$ to $\mathcal{A}$. When $\mathcal{A}$ produces an output, $\mathcal{B}$ uses this as its own output. This completes the simulation.

To complete the proof, we must show that if $f_\cdot(\cdot)$ is modeled as a random oracle, the simulated distribution provided to $\mathcal{A}$ is computationally indistinguishable from the **Real** experiment on the same inputs. This implies that the distribution of $\mathcal{B}$'s output must in turn be computationally indistinguishable from that of $\mathcal{A}$ in the **Real** experiment.

Our proof proceeds via a series of hybrids. The first hybrid represents the **Real** experiment, while the final hybrid is distributed as in the **Ideal** experiment. We will define $\mathsf{Adv}[i]$ to be the quantity $|\mathbf{Pr}\,[\,\mathcal{Z}(\mathbf{Hybrid\ i}(\mathcal{A}, \ell, \mathcal{I}, N, \mathcal{U})) = 1\,] - \mathbf{Pr}\,[\,\mathcal{Z}(\mathbf{Real}(\mathcal{A}, \ell, \mathcal{I}, N, \mathcal{U})) = 1\,]|$.

Hybrid 0. This hybrid implements the experiment $\mathbf{Real}(\mathcal{A}, \lambda, \ell, \mathcal{I}, N, \mathcal{U})$. Clearly $\mathsf{Adv}[0] = 0$.

Hybrid 1. This hybrid modifies the above hybrid as follows: for all $f(\cdot, \cdot)$ oracle queries $\mathcal{A}$ makes *prior* to receiving W, if the oracle query has the form $(k, \cdot)$, then abort the experiment and output $\perp$.

We observe that k is random and not in $\mathcal{A}$'s view prior to receiving W. Thus

[5] If the oracle has already been queried by $\mathcal{A}$ (and thus implicitly defined at this point) prior to this stage of $\mathcal{B}$'s operation, $\mathcal{B}$ aborts and outputs $\perp$.

if $\mathcal{A}$ makes q queries, the probability of abort is at most $q \cdot 2^{-\lambda}$. This bounds $\mathsf{Adv}[1] - \mathsf{Adv}[0] \leq q \cdot 2^{-\lambda}$.

Hybrid 2. This hybrid modifies the above hybrid as follows: it randomly samples the coefficients $a_0, ..., a_{M-1}$ that define $P(\cdot)$ and programs the random oracle such that $\forall i \in [0, N)$, $f_k(i) \equiv P(i) + I_i \mod \ell$.

Let us implicitly define $\mathcal{I} = (I'_0, \ldots, I'_{M-1})$, and $\mathcal{T} \setminus \mathcal{I} = (I'_M, \ldots, I'_{N-1})$, and define the corresponding locations of these transactions in $\mathcal{T}$ as $(j_0, \ldots, j_{N-1})$. We make the following observations:

1. The distribution of $(a_0, ..., a_{M-1})$ is identical to that of the previous hybrid. This is because in **Hybrid 1** it holds that (1) each $f_k(\cdot)$ is uniformly distributed in $\mathbb{F}_p$, and (2) for a given $\mathcal{I}$ and $\forall i \in [0, m)$ the value $P(i)$ is uniquely determined by $f_k(\cdot)$, thus (3) each pair $(f_k(i), P(i))$ is equally probable.This implies that the distribution of these pairs is identical in this hybrid and the previous hybrid. Because there is exactly one unique polynomial for each set of M such points, and the points are uniformly distributed in $\mathbb{F}_p^M$, then the coefficients in **Hybrid 1** must also be distributed uniformly in $\mathbb{F}_p^M$. Thus the coefficients are distributed identically in the two hybrids.

2. Similarly, each distinct tuple of coefficients $(a_0, ..., a_{M-1})$ uniquely defines a distinct tuple $\hat{P} = (P(j_0), \ldots, P(j_{M-1}))$. Because the coefficients are sampled uniformly in this hybrid, then the tuple $\hat{P}$ is distributed uniformly in $\mathbb{F}_p^M$. Thus $\forall i \in [0, M)$ it holds that $f_k(j_i) \equiv P(j_i) + I'_{j_i}$ *mod* ℓ is uniform. This is identical to **Hybrid 1**.

3. For every index $i \in (j_M, \ldots, j_{N-1})$ (*i.e.,* the indices not in $\mathcal{I}$, every index I'_{j_i} in **Hybrid 1** is uniformly sampled from $\mathbb{Z}_\ell$. Provided that p is substantially

larger than ℓ (by a large factor) then each point $f_k(j_i)$is also distributed (nearly) uniformly and indistinguishably from **Hybrid 1**.

As a consequence, all values provided to $\mathcal{A}$ and all oracle queries are distributed identically to **Hybrid 1**. This bounds bounds $\mathsf{Adv}[2] - \mathsf{Adv}[1] \leq \frac{1}{p/\ell}$.

By summation over the hybrids we have that $\mathsf{Adv}[2] - \mathsf{Adv}[0] \leq q \cdot 2^{-\lambda} + \frac{1}{p/\ell}$, and therefore if $\frac{1}{p/\ell}$ is negligible then $\mathcal{B}$'s success probability is at most negligibly different from that of $\mathcal{A}$. □

4.5 Duplicates and Alternative Distributions

The scheme we presented above serves as a useful first step. However, two major problems remain. First, the transaction list $\mathcal{T}$ contains duplicates, which may reduce the effective size of the anonymity set. Second, while our above technique works well for uniform distributions, many desired applications may use alternate distributions. We discuss both issues below.

4.5.1 Duplicates

It is important to note that the scheme above offers only a probabilistic guarantee for the actual number of unique transactions in $\mathcal{T}$. There is a chance that duplicate transactions will occur in this list, resulting in a (unique) set of size less than N. We discuss several ways to handle this.

4.5.1.1 Expected number of unique transactions

The simplest strategy is to accept the existence of duplicate transactions and instead focus on the expected number of unique transactions for a transaction list of size N. We can compute this with the following formula:

$$\ell(1 - (\frac{\ell - 1}{\ell})^N)$$

If we use concrete numbers based on Monero with $\ell = 4,000,000, N = 1,000$, the expected number of unique transactions is 999.88. Based on this, it seems the risk of duplicate transactions is fairly low. However, we can do better if we truly want to minimize this risk.

4.5.1.2 Resampling

To eliminate duplicates entirely, we can resample $\mathcal{T}$ until we obtain N unique transactions. We can bound the number of attempts needed to minimize the chance of obtaining duplicates to 2^{-50} with the following formula, where r is the number of resamples.

$$r = \frac{-50\log(2)}{\log(1 - \frac{\ell!\ell^{-N}}{(\ell-N)!})}$$

Using concrete numbers ($\ell = 4,000,000, N = 1,000$), then we find after 17 resamples the probability of duplicates is within this bound. This may have a modest impact on the security reduction.

4.5.1.3 Oversampling

A final strategy is to actually sample N' values so that we obtain N unique transactions with overwhelming probability. We do this by computing the required value of N' such that the probability of obtaining fewer than N unique transactions is bounded by 2^{-50} using the following formula:

$$N' = (N+1) - \frac{\log(\frac{2^{50}}{\sqrt{2*\pi}})}{(\log(1/\ell))}$$

Using concrete numbers ($\ell = 4,000,000, N = 1,000$), then we find that if we set $N' = 1,004$ we will obtain N unique transactions with overwhelming probability.

$F_{MONERO}(\mathcal{I} = \{I_0, \ldots, I_{M-1}\}, N)$

1 : For $i = 0$ to $N - M + 1$:
2 : Sample a random $r \in \mathbb{Z}_{2^{53}}$.
3 : Compute $\Gamma_i = \sqrt{\frac{r}{2^{53}}} \cdot \ell$.
4 : Construct sorted set $\{T_0, \ldots, T_{N-1}\}$ of $\Gamma_0, \ldots, \Gamma_{N-M+1}, I_0, \ldots, I_M$.
5 : **return** $\mathcal{T} = \{T_0, \ldots, T_{N-1}\}$

Figure 4-2. Monero sampling technique

4.5.2 Alternative Distributions

Some currencies, including Monero, use sample cover transaction outputs according to a non-uniform distribution. This strategy is designed to produce cover traffic that more closely resembles the distribution of real transaction outputs chosen by users. At a basic level, Monero achieves this goal by using a distribution that is biased against older transactions. Nominally this distribution is referred to as a triangular distribution, however it has differences from a triangular distribution so that it has further bias against older transactions. At a low level, Monero samples random numbers uniformly, and then transforms them to the appropriate distribution. We provide a outline of the Monero sampling technique in Figure 4-2.

Given this distribution, we will demonstrate how to apply the technique from Figure 4-1 to this distribution. In the sampling method we modify the y_i values to be computed in the following way:

$$y_i = f_{N,\ell}(k, j_i) - \frac{I_i^2}{\ell^2} \cdot 2^{53} \ mod \ \ell$$

And compute the T_i values as follows:

$$T_i = \sqrt{\frac{f_{N,\ell}(k, i) - P(i)}{2^{53}}} \cdot \ell \ mod \ \ell$$

Similarly we modify the recover algorithm to compute T_i as above.

For distributions such as the one above which utilize uniform sampling in their method, the conversion to our scheme is fairly straightforward. We simply replace the uniform sampling step with our programmable hash function. However, this is a special case and a more general strategy may be needed. In those situations we can simply make use of Inverse Transform Sampling. The general idea is that for a distribution X, with CDF F_X, we simply do the following: If u is the result of our uniform sampling technique, we solve for x s.t. $F_X(x) = u$. This implies that using our uniform sampling technique. we can construct an RSS for any distribution.

4.6 Integration with Specific Cryptocurrency Protocols

In this section we provide concrete examples of how this technique can be implemented within specific cryptocurrencies. Here we focus on Monero (RingCT) and ByteCoin (CryptoNote), due to the fact that these are currently the most widely-used mixing cryptocurrencies and protocols. For each system we provide bandwidth cost estimates for using our approach in that system, and compare with the cost of the approach currently used by the system.

Remark: In the following analysis we focus only on the description of $\mathcal{T}$ as it appears within a transaction. The analysis below omits all additional data that real currencies embed in their transactions, including (notably) cryptographic proofs. In current versions of mixing currencies, *e.g.,* Monero, the size of these proofs grow linearly in N, with Monero requiring an approximately 13KB transaction for $M = 2, N = 10$. Future proof systems [14, 94, 100] offer bandwidth that grows logarithmically in N or better. For example, Bulletproofs [14] may offer proof sizes under 3KB.

4.6.1 Overview of Protocols

We now describe the cryptocurrency protocols, and specifically the approach each uses to encode $\mathcal{T}$.

ByteCoin. Bytecoin [15] is one of the earliest mixing protocols, and is based on the CryptoNote protocol [85]. ByteCoin uses a 1-out-of-N approach for constructing $\mathcal{T}$. If a transaction references multiple real transaction outputs, ByteCoin will sample multiple $\mathcal{T}$. Each $\mathcal{T}$ is differentially encoded within the transaction, *i.e.,* for each $i \in [1, N)$ the transaction encodes $I_i - I_{i-1}$.[6]

Because CryptoNote does not hide payment values, cover traffic outputs are sampled uniformly from the subset of previous transation outputs *that have the same currency value* as the real transaction output being obscured. At the time of writing, the largest such subset (for outputs of value 100 BCN) gives us $\ell = 2,000,000$, and we use this for our analysis. In our simulations below, we use an estimate for the size of each ByteCoin description of $\mathcal{T}$ by computing the expected size of N uniformly-sampled indices when differential encoding is applied. Note that in ByteCoin, values of $M > 1$ are encoded using M *distinct* transaction lists $\mathcal{T}$, one for each real transaction. For simplicity, we compute our ByteCoin simulations only for $M = 1$.[7]

Monero. The Monero currency [67] is based on the newer RingCT protocol [70]. Like ByteCoin, Monero uses a 1-out-of-N approach for constructing $\mathcal{T}$ (for values of $M > 1$ this results in M distinct lists $\mathcal{T}$ being sampled and encoded within the transaction). Unlike CryptoNote, RingCT hides transaction values. Because transaction outputs do not need to be bucketed by like value, this gives a larger available set of previous transactions for a spender to use as cover traffic: at the time of writing, approximately

[6] These are encoded using a standard VLQ integer encoding. The low-order 7 bits of each byte encode data, and the MSB acts as a flag indicating whether there are additional bytes to come.

[7] To evaluate the case where $M > 1$, one can simply take the measurements for $M = 1$ and multiply the resulting description size by M. Of course this is somewhat unfair to Monero, given that a more efficient version of the protocol could simply encode multiple real transactions within a single set $\mathcal{T}$.

$\ell = 4,000,000$. Monero encodes the transaction indices of $\mathcal{T}$ using the same differential encoding as ByteCoin.

As we discussed in §4.5, Monero samples cover traffic using an (approximate) triangular distribution.[8] For our simulations, we estimated the size of a Monero encoding of $\mathcal{T}$ by running Monte Carlo simulations with $1,000$ trials and computing the average size. Monero uses the same encoding process as ByteCoin for values of $M > 1$, and so again we consider our results only for the case of $M = 1$.

4.6.2 Simulation Results

For this simulation we assume our RSS scheme uses a 128-bit key k and a 64-bit prime p.[9] Larger values for these parameters are possible, and will have a predictable impact on our efficiency.

Figure 4-3 provides an clear visualization of how our scheme compares to the existing approach used in Monero and ByteCoin. For both currencies, we plot how varying N impacts the size in bits of the transaction list. Because the size of our RSS compact description depends on M, we offer plots for several values of M.

Monero. As expected, the "current approach" Monero encoding grows linearly with the number of transactions included in the transaction list. When only one legitimate transaction is included ($M = 1$), we see that for $N \geq 9$ our scheme produces a smaller description than the current Monero approach. Even as we increase M to larger values $(2, 5)$, our scheme still outperforms Monero's current approach at reasonably small values of N.

[8] In practice, Monero's sampling is more complicated. Monero uses two sets, one of "recent" transaction outputs, and another of "all previous" transaction outputs, and ensures that at least 50% of the resulting transactions are in the recent set. For simplicity of exposition, we simulate only the simple process of sampling from the set of all previous transactions, although our results can easily be adapted to Monero's more complicated sampling process.

[9] Our results do not include the description of p or ℓ, as we assume that ℓ is indicated elsewhere in the transaction (as it is in Monero and ByteCoin) and that p can be identified deterministically given ℓ.

N	RSS ($M = 5$)	Monero	ByteCoin
1,000	.06 kB	1.97 kB	5.94 kB
10,000	.06 kB	16.59 kB	55.4 kB
100,000	.06 kB	103.17 kB	497.86 kB

Table 4-I. Anonymity Costs of each encoding approach for large N

ByteCoin. The "current approach" of ByteCoin is similar in appearance to Monero, although it differs subtly in the slope of the line. Again, our scheme proves superior to ByteCoin's current approach, though at somewhat smaller values of N in comparison with Monero. Here, our scheme offers better cost for $N \geq 4$. Additionally, increasing M only has a small impact on the comparative efficiency of our scheme.

The reason for the steeper slope of the "current approach" in the case of ByteCoin is due to the fact that the currency samples transactions from a uniform distribution. This causes the transaction indices to be more evenly spaced, and thus all require roughly the same number of bits. Since Monero has a preference for recent transactions, many of the included transactions will be numerically close and will therefore produce more compact differential encodings.

In both cases, our scheme outperforms current approaches at even relatively small values of N. Additionally, innovations such as new proof techniques [94, 100] promise dramatic increases to what is considered a practical value for N. To illustrate the potential cost savings of our approach at larger values of N, we provide the following concrete numbers (for an exemplary $M = 5$) in Table 4-I.

These bandwidth comparisons illustrate the potential for our scheme to offer significant cost savings in transaction bandwidth. For comparison, the current average *total* transaction size in Monero (including all proofs and metadata) as of this writing is approximately 13kB [103].

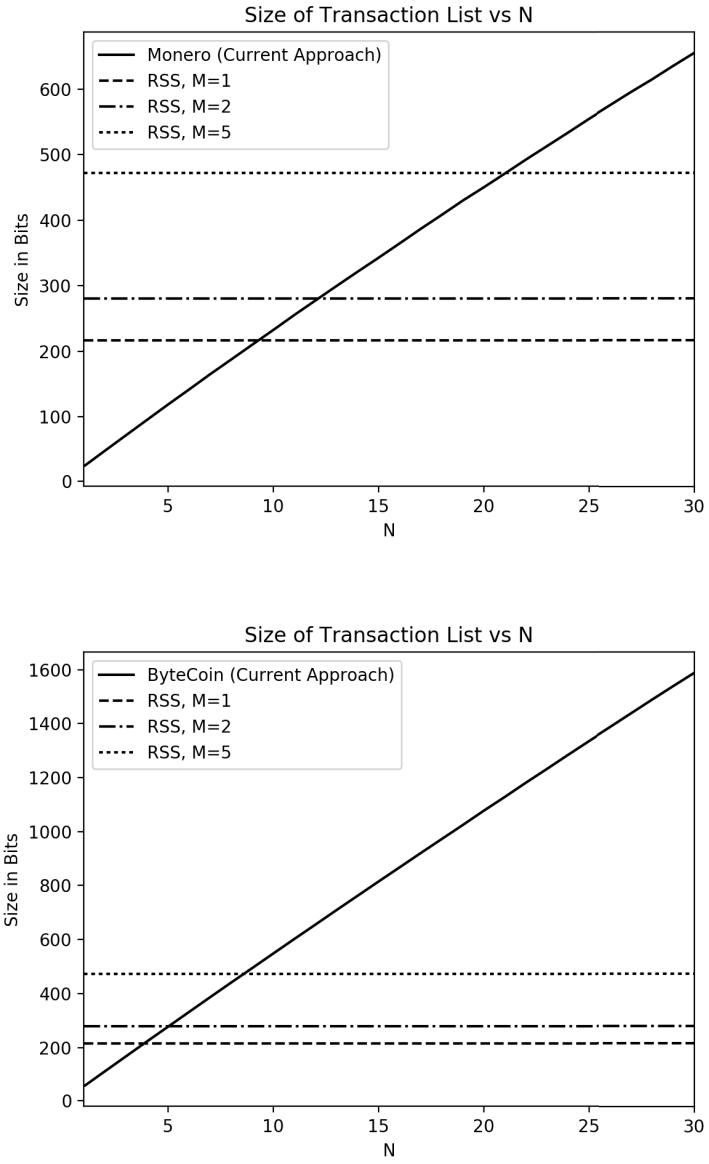

Figure 4-3. Size of the transaction list $\mathcal{T}$ in bits for two popular cryptocurrencies, Monero (top) and ByteCoin (bottom). "Current approach" indicates the representation of the transaction list as a function of N (the number of transactions on the list). Note that this value is (mosty) independent of M, so we do not illustrate different values of M. "RSS" indicates the cost of the compact description W for different values of M, N. For ByteCoin we use $\ell = 2,000,000$ and for Monero we use $\ell = 4,000,000$.

4.7 Other Applications

Beyond the significant improvements offered in the area of mixing cryptocurrencies, RSS has the potential for extensions to a more diverse set of applications. Generally speaking, the idea behind our scheme is to enable the insertion of desired values into a distribution, without revealing where these insertions took place. In this section we offer two examples of how to leverage RSS in other areas.

Anonymous Communication. The general concept of "mixing" is applicable to many anonymous communications systems. In particular, Rivest *et al.* originally proposed using ring signatures to add plausible deniability to email communications [82]. Just as the true input to a cryptocurrency transaction can be hidden by adding cover inputs, the true sender of a message can be hidden by mixing with other identities. If all user public keys are posted on some publicly accessible bulletin board, then we can use RSS to efficiently sample this data according to some distribution. Thus, senders would be able to construct large, yet compact, anonymity sets for themselves. Of course, just as in the cryptocurrency case, auxiliary data such a cryptographic proof will determine the total bandwidth overhead.

Client-Server Puzzles. A somewhat different application of RSS comes from Client-Server Puzzles [49], which generally involves one party requiring another party to complete some computationally intensive task. A common example is searching the domain of some hash function for an output that satisfies a particular requirement. RSS allows the puzzle creator more control over the search space. They might choose to place the solution at a specific position or modify the difficulty of the puzzle by changing the number of solutions present. Alternatively they could introduce trapdoors into the puzzle, by placing a solution at an index that an honest client can compute using their shared secrets.

This is far from a comprehensive list of the applications of RSS, and a future

extension of this work is to explore and enumerate the possibilities.

4.8 Related Work

4.8.1 Anonymity for cryptocurrencies

In addition to the anonymity protocols mentioned earlier, a separate line of works seek to increase anonymity by Bitcoin by allowing users to interactively mix transactions (e.g. CoinJoin [61], CoinShuffle, CoinSwap). A separate line of work [36, 40] examines *off-chain* private payments.

4.8.2 Improved WI and ZK proof techniques

A different line of work examines the efficiency of privacy-preserving protocols such as ring signatures, which are widely used in cryptocurrencies. For example, Dodis *et al.* [26] proposed a constant-sized[10] RSA-based ring signature (using an accumulator due to Camenisch and Lysyanskaya [17]) in the random oracle model. Using a new proof system in the discrete log setting, Groth and Kohlweiss recently proposed a concretely efficient technique with $log(n)$-sized signatures [37] in the random oracle model. Most recently, Malavolta and Schröder proposed an efficient *constant-sized* group signature in the CRS model based on zkSNARKs [58].

4.8.3 Programmable hash functions

Many works use programmable hash functions. Hofheinz and Kiltz [41] offer a canonical paper on the technique, although their proposals are not precisely suited to our application. A vast number of works (including for example [9, 42, 99]) employ these techniques. A noteworthy feature of some schemes similar to ours, (*e.g.,* the

[10]For convenience we ignore the security parameter. In practice, such "constant-sized" schemes have bandwidth $O(\lambda)$.

Hohenberger-Waters signature [42]) is that those schemes require *statistical* properties from underlying hash function f, and thus achieve security in the standard model even when the hash function is a PRF with a non-secret seed. Developing an RSS in the standard model is an interesting open problem.

4.9 Conclusion

In this work we described a new approach to describing transaction sets in deployed mixing cryptocurrencies such as Monero and ByteCoin. We believe that our technique is promising and can offer substantial bandwidth improvements when the total size of the transaction set is large. Our work leaves several open questions. In particular, we believe that there may be many other applications for this technique. Additionally, we desire a scheme that offers security without relying on random oracles.

Chapter 5

Detecting Man-in-the-Middle via Network Latency and Reverse Turing Tests

5.1 Technical Overview

Our contributions in this work follow the ideas of previous forced latency protocols such as Interlock [81] and Zooko's "Defense Against Middleperson Attacks" protocol [101]. The basic protocol works by leveraging the idea of *contributory key agreement*. Such protocols extend standard key agreement protocols with a strong guarantee that no party can predict the outcome of the key exchange with an honest counterparty. Thus, if a MitM is present that inserts itself into the key agreement protocol, it cannot cause both honest parties to arrive at the same key. Thus, in the MitM case, the honest parties will obtain mismatched keys.

The objective of a forced latency protocol is to exploit this key mismatch in order to reveal the presence of an attacker. This is done by designing a protocol that forces an MitM attacker to either (1) dramatically increase the latency of the communication with a participant's counterparty, or (2) take on the role of the counterparty, and thus successfully emulate that party's communications. The assumption that underlies these protocols is that, in either case, the communicating parties will be able to

conclude that their conversation is being tampered with.

In the protocols described above, there are two aspects that are required in order to achieve this goal. First, each party must ensure that each message is transmitted under a unique key that is tightly bound to the party's encryption key. Next, is the forced latency piece: parties do not transmit a message for immediate consumption; instead, they transmit a *committing encryption* of the message and then, after waiting for some set amount of time, transmit an opening key for this ciphertext. This poses a challenge for a passive attacker, who must decrypt and re-encrypt the message when relaying between parties, resulting in a doubling of the latency. Of course, an active attacker can simply take on the role of the counterparty — and need not relay the real counterparty's messages — but this requires the attacker to emulate the counterparty's message style prior to seeing their messages. As shown in figures 5-1 and 5-2, unless the MitM wants to increase the latency noticeably it must either reveal the key mismatch or forge new messages and commitments.

While the basic idea behind forced latency protocols is fairly elegant and simple, we found that formalizing these ideals led to some non-trivial technical problems

The acknowledgement problem. One of the most common components not fully fleshed out by existing protocols is how a party actually detects the MitM's presence. Informally, it is assumed that if Party A started the protocol, then the receipt of Party B's message will be used to compute latency. However, this raises the question of what this measured latency is compared against. Additionally, for the most robust definition we would also like to know whether Party B actually received our messages and when they were received. However, these acknowledgements cannot be sent over in the clear as that would allow the MitM to tamper with them. Thus, the forced latency protocol must be modified so that these acknowledgements can also be sent over in a protected manner.

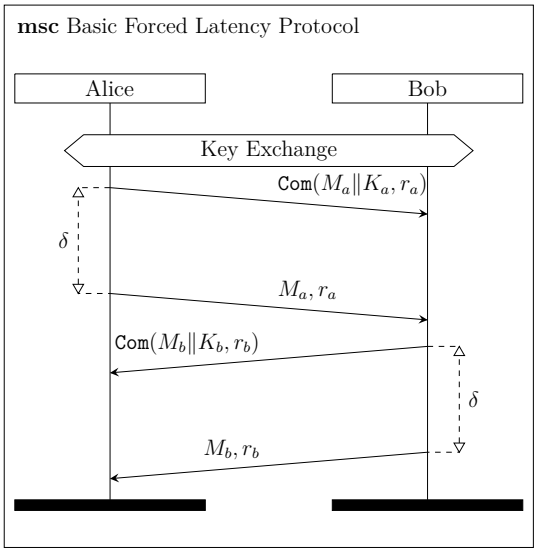

Figure 5-1. Illustration of basic forced latency protocol

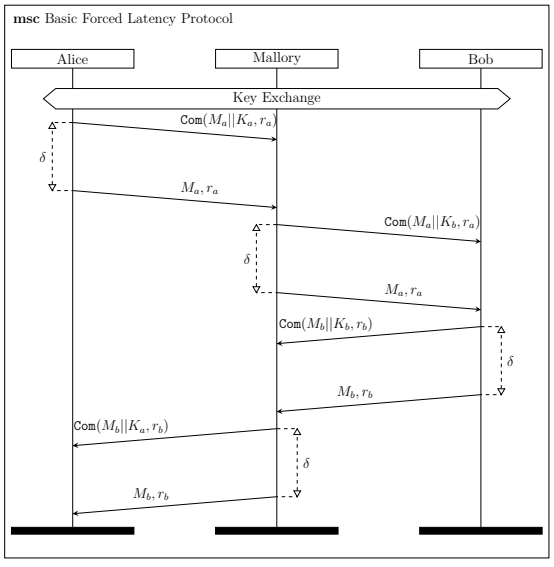

Figure 5-2. Illustration of forced latency protocol with (passive) MitM present

Forgery. A third option an MitM has outside of adding more latency or doing nothing is forging messages. Some existing protocols attempt to prevent this by introducing the idea of "semantic irregularities". This is the idea that the messages being exchanged are too high entropy for the MitM to forge without being exposed. However, there was never a formal definition introduced for this notion nor a clear idea on how to achieve this property. This raises the question of how to capture this notion in a way that is compatible with provable security.

Deepfakes and Emulated Human Behavior. Beyond the limitations already mentioned for text based human authentication systems, the recent rise of LLMs such as chatGPT [71] have further complicated the problem of identifying a human from text conversation alone. This problem continues into the audio/video realm as well with the rise of deepfakes [60]. The development of these sophisticated models for emulating human behavior have led to some asking whether the turing test is even relevant anymore [10].

The key threats of these models are the ability to produce full machine produced forgeries of human behavior as well as to allow for human adversaries to better imitate other parties.

Our approach in this work is to leverage the natural capabilities of human beings to “authenticate” human beings to produce a strong authentication protocol for cryptographic interactions. Naturally, this approach implicitly depends on the ability of human beings to distinguish true interactions from “deepfake” interactions.

We note that while deepfakes continue to get more advanced, in practice these work best on videos that are mainly static [65, 92] and are not well suited to interactive settings [10]. In the near term, this indicates that for our setting of video calls, these techniques are still detectable.

However, with the rapid development of deepfake technology, it is important to

recognize that these imitations will eventually reach a level where the forgery is not obvious to the human eye. At the same time, our capabilities to detect these types of emulations of human behavior are also rapidly improving, both in the text [66] and deepfake [52, 60, 102, 104] settings. Thus in the longer term, humans with additional detection tools may be able to rule out such attacks.

To account for this, in our formalization of human authentication we include the possibility of humans leveraging these tools to detect forged video (See 5.2.1). Ultimately our system cannot survive an environment in which humans cannot authenticate other human beings. We view our work as an investigation into what is possible in today's environment.

Improving the "Real-Time"-ness of the Protocol. Forced latency protocols marry two somewhat contrary notions. The latency measuring properties of these protocols necessitate that the protocol is used for real-time communication. However, the protocol requires the addition of artificial delays. As a result, most existing protocols are turn-based. Parties take turns sending messages and can only reply when it is their turn to do so. This is not a common pattern for real-time communication, so it raises the question of whether a forced latency can allow for simultaneous communication between parties. For example, most real-time communication protocols tend to have simultaneous streams of audio or video from both parties. It stands to reason that the closer the protocol is to an in-person conversation the easier it will be to detect irregularities. While having some delay in a conversation may cause annoyance, we believe it to still be superior to having to wait your turn to talk. In a sense, turn-based communication is simply real-time communication with severe latency already.

Party Synchronicity. In formalizing these protocols an important question that arose is how to prevent the parties from falling out of sync or having differing

information about the presence of an MitM. Existing work generally only had one party learn of the presence of the MitM at a time, so its possible that a party doesn't realize it is vulnerable. Furthermore, the risk of the MitM pushing the parties out of sync to alter their latency measurements is not formally analyzed. Ideally, the adversary should not be able to act without both parties detecting it immediately.

Amount of delay. The exact process of choosing a latency was also not well specified. Selecting the added latency δ appropriately is critical to the functioning of our protocols, it must be large enough that the presence of the delay is clearly detectable even against the nosiness of real networks. As the parameter δ cannot be set dynamically as that may allow the MitM to manipulate it to hide their presence, we need to capture a picture of what expected latencies between different regions are like. MitM attacks are infrequent and should not be able to significantly affect these measurements of expected latency. Then in practice, we can map an appropriate δ based on the expected latency. That way we can ensure a suitable δ is chosen. We discuss this in section 5.5.4 in our examination of TURN servers in different regions and the associated latencies. These results provide some indication of a future method of heuristically selecting the latency based on location in a way that is still robust to attempts by the adversary to manipulate the value.

For our proof of concept, we selected a value of δ which we found in practice would be the dominant contribution to latency in most circumstances. We note that if users are willing to tolerate latency beyond the normally acceptable threshold (>400ms)[1] an applicable fixed value of δ can always be found.

In our experiments detailed in section 5.5.2 we note that once the average latency in a 10 second window is larger than 2*δ, we assume the presence of an MitM. As we discussed, δ is chosen so that the presence of forced latency delay is clearly detectable.

[1]Cisco found that a one-way delay of 200ms was the threshold before quality noticeably declines [77]. We extrapolate from this to arrive at a threshold for roundtrip delay

Depending on how strict of a MitM detection strategy is desired, detection could alternatively be based on majority vote of multiple measurement windows or as soon as any measured latency is $2^*\delta$.

Resolving these issues required significant adjustments to the design of a forced latency protocol which we discuss in more depth in section 5.3. Taken all together we arrive at LATENT, a realistic, formally defined and proven Forced Latency protocol.

5.2 Definitions

Our security definition is motivated by the *authenticated and confidential channel establishment* (ACCE) protocol from [44]. This protocol provides a powerful framework for evaluating the security of an authenticated key exchange. However, some modifications are necessary since the original ACCE definition was motivated with TLS in mind and we are dealing with a stronger adversary and an unauthenticated key exchange. We will refer to our definition as *human authenticated secure communication* (HASC).

5.2.1 The Turing Test

A goal of this paper is to provide a formal representation of the use of 'human authentication'. Informally, this refers to leveraging an actual human user, rather than just their system when executing a protocol.

As the rest of these protocols can be made to sit on firm theoretical bases, our objective is to encapsulate this informal idea into a single primitive. That way, just as other building blocks can reduce to cryptographic assumptions, the human aspect can too be reduced to some hard assumption. This modularity would also simplify replacing this component if an assumption is found to be faulty without having to rework the rest of the protocol.

Since these protocols are leveraging humans to perform tasks that are presumed difficult for computers to do, it makes sense to tie this primitive to Hard AI problems. Breaking this primitive should be just as hard as being able to solve such a problem.

The gold standard of such a problem is the concept of the Turing Test. The Turing Test is an interactive protocol between a human challenger and a potentially non human adversary. The human can send as many challenges as it wishes and once it is satisfied it will output accept or reject. Only a human adversary should result in an accept.

To account for deepfakes and other advances in human emulation, we augment the basic Turing Test to allow for humans to make use of additional resources. We provide the human challenger optional access to a detection tool that can take messages from the adversary and check whether they were produced by an ML model. Such tools are available [52] and hamper the use of algorithmic attempts to imitate a human. Critically, the final decision still comes down to the human, but these tools can help inform their decision. An AI beyond the detection capabilities of a human with these resources would indicate that humans are no longer able to virtually authenticate each other. Any algorithmic turing machine adversary should only succeed with negligible probability.

In formalizing the game, we had the challenge of keeping the overall properties of the Turing Test from being too specific to our setting, while still exposing the correct interface needed for our security proofs.

5.2.1.1 Formal Definition

We formalize the idea of a Turing Test for authentication as follows. Let $\mathcal{H}$ be a human participant who will communicate with a potentially nonhuman adversary $\mathcal{A}$. To setup the game $\mathcal{H}$ receives some auxiliary system parameters $\mathtt{pp}$ which detail any necessary context for the game. They also have oracle access to a detection machine

$\mathcal{D}$. The protocol proceeds with $\mathcal{H}$ and $\mathcal{A}$ exchange messages. We denote the i-th message sent by party $\mathcal{P} \in \{\mathcal{H}, \mathcal{A}\}$ as $m_i^{\mathcal{P}}$. The transcript $< \mathcal{H}, \mathcal{A} > (t) \rightarrow \tau$ at time t is the tuple $(\{m_i^{\mathcal{H}}\}_{i=1}^{p}, \{m_i^{\mathcal{A}}\}_{i=1}^{q})$ where p (resp. q) is the number of messages sent by party $\mathcal{H}$ (resp. $\mathcal{A}$) by time t. $\mathcal{H}$ can query $\mathcal{D}$ on any message it receives and will receive a bit b that is 1 if the message was the output of an ML model and 0 otherwise. On any transcript τ, $\mathcal{H}(\tau)$ will output $d \in \{\texttt{accept}, \texttt{reject}, \texttt{undecided}\}$. We define the decision time t_d and decision transcript τ_d such that τ_d is the transcript at time t_d and $\mathcal{H}(\tau_d)$ outputs either `accept` or `reject`.

Definition 5.1. *We say that a Turing Test is (t, ϵ)-authenticating if for any nonhuman $\mathcal{A}^*$ there exists a negligible function $\nu(\cdot)$ such that it holds that*

$$Pr\left[\texttt{accept} = \mathcal{H}(\tau) \middle|\ \tau \leftarrow < \mathcal{H}, \mathcal{A} > (t)\ \right] \leqslant \nu(\cdot)$$

Where a minimum of ϵ messages did not originate from a human participant.

5.2.2 HASC Protocols

In this section we provide an overview of HASC protocols and their security. First we describe the model at a high level and then detail the formal model and definitions.

5.2.2.1 Overview

An HASC protocol is a protocol executed between two parties. The protocol consists of three phases, called the 'pre-accept' phase, 'auth' phase, and the 'post-accept' phase.

Pre-accept phase. In this phase a 'handshake protocol' is executed and a session key k is established. This phase ends, when both communication partners reach an `accept-state` (i.e. $\Lambda_{pre} =$ '`accept`').

Auth phase. In this phase an 'a posteriori' authentication protocol is executed

through the invocation of an instance of the Turing test. The exact behavior of the human challenger will be a function of the protocol. This phase ends, when both communication partners reach an `auth-state` (i.e. Λ_{auth} = '`accept`').

Post-accept phase. This phase is entered, when both communication partners reach an `auth-state`. In this phase data will be transmitted, encrypted and authenticated with key k.

At a high level, these phases capture the differing states of security provided to the participants in an *a posteriori* authenticated key agreement protocol. First, participants negotiate a key and use it for encryption. At this stage the key is considered to be insecure. Then, they perform some additional protocol to verify the identity of their fellow participant. If the Authentication succeeds, then the key is considered to be secure. Finally, this key can be used to provide a confidential communication challenge.

In order to model this setting, a somewhat involved execution environment must be defined. This environment is meant to capture the capabilities of an Adversary. The Adversary is specified to facilitate all communication in the environment so to capture the large extent of its control over all aspect of the protocol. At the high level, the adversary interacts with protocol participants in the following ways:

Send queries. These queries deliver a message to a participant and return a response. Using these queries allows an adversary to carry out communication between participants as well as inject its own messages.

Corrupt queries. The adversary is additionally able to receive the session key or long term keys of parties. This captures the idea of forward secrecy and post compromise security.

Encryption queries. Once the sessions keys are established and the Post-accept phase begins, an adversary must use these queries to transfer messages between parties.

For a HASC protocol to be secure, we want to require either that the Auth phase fails or that messages in the post-accept phase are hidden to the adversary in the event that an adversary interferes in the pre-accept phase. That is to say that we should be able to condition on the consistency of messages sent and received in the pre-accept phase and show that in either case there is a challenge the adversary will fail at.

We do this by separating the security into a pair of games. There is the Authentication game where an adversary must interfere during the session key establishment in the pre-accept phase but not have the participants detect the interference during the auth phase. Then there is the Encryption Game, where session keys should be shown to be secure if the session keys are honestly established.

Clearly, if an adversary cannot win at either game we will have an authenticated channel by the end of the protocol.

5.2.2.2 Execution Environment

5.2.2.2.1 Pre-accept Phase Consider a set of parties $\{P_1, ..., P_\ell\}$, $\ell \in \mathbb{N}$, where each party $P_i \in \{P_1, ..., P_\ell\}$ is a (potential) protocol participant and has a long-term key pair (pk_i, sk_i). (Note that unlike the standard AKE model [2], we do not assume that these public keys are known to the other parties.) (Actually, the public keys should be provided by a potentially malicious party) To model several sequential (We feel its a natural assumption that a human can only be party to one execution at a time) of the protocol, each party P_i is modeled by a collection of oracles $\pi_i^1, ...\pi_i^d$ for $d \in \mathbb{N}$ and a human $\mathcal{H}_i$. Each oracle π_i^s represents a process that executes one single instance of the protocol. All oracles $\pi_i^1, ...\pi_i^d$ representing party P_i have access to the same long-term key pair (pk_i, sk_i) of P_i. Moreover, each oracle π_i^s maintains as internal state the following variables:

- $\Lambda_{pre} \in \{\texttt{accept}, \texttt{reject}\}$.

- $k \in \mathcal{K}$, where $\mathcal{K}$ is the keyspace of the protocol.
- $\Pi \in \{1, ..., \ell\}$ containing the intended communication partner, i.e., an index j that points to a public key pk_j used to perform authentication within the protocol execution.
- Variable $\rho \in \{\mathsf{Initiator}, \mathsf{Responder}\}$.
- Some additional temporary state variable st (which may, for instance, be used to store ephemeral Diffie-Hellman exponents or the transcript of all messages sent/received during the Handshake).
- bit $b_i^s \xleftarrow{\$} \{0, 1\}$, chosen at random at the beginning of the game.

The internal state of each oracle is initialized to $(\Lambda_{pre}, k, \Pi, \rho, st) = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$, where $V = \emptyset$ denotes that variable V is undefined. Furthermore, we will always assume (for simplicity) that $k = \emptyset$ if an oracle has not reached `accept-state`(yet), and contains the computed key if an oracle is in `accept-state`, so that we have

$$k \neq \emptyset \iff \Lambda = \text{`}\texttt{accept}'.$$

An adversary may interact with these oracles by issuing the following queries:

- $\mathsf{Send}^{\mathsf{pre}}(\pi_i^s, m)$: The adversary can use this query to send message m to oracle π_i^s. The oracle will respond according to the protocol specification, depending on its internal state, except that it replies with an error symbol $\perp$ if oracle π_i^s has state $\Lambda_{pre} = \text{`}\texttt{accept}'$. (Send-queries in an `accept-state` are handled by the $\mathsf{Send}^{\mathsf{auth}}$ and Decrypt-queries below).

  If the attacker asks the first Send-query to oracle π_i^s, the the oracle checks whether $m = \top$ consists of a special 'initialization' symbol $\top$. If true, then it sets its internal variable $\rho :=$ Initiator and responds with the first protocol

message. Otherwise it sets $\rho :=$ Responder and responds as specified in the protocol.

The variables Λ, k, Π, st are also set after a Send-query. When and how depends on the considered protocol.

- Reveal(π_i^s): Oracle π_i^s responds to a Reveal-query with the contents of variable k. Note that we have $k \neq \emptyset$ if and only if $\Lambda =$ '`accept`'.

- Corrupt(P_i): Oracle π_i^1 responds with the long-term secret key sk_i of party P_i. If Corrupt(P_i) is the τ-th query issued by $\mathcal{A}$, then we say that P_i is τ-*corrupted*. For parties that are not corrupted we define $\tau := \infty$.

5.2.2.2.2 Auth phase For the auth phase each oracle π_i^s keeps an addition variable Λ_{auth}. During this phase, the human $\mathcal{H}_i$ representing party P_i conducts a Turing test with $\mathcal{H}_\Pi^*$, to assess whether it is communicating with the human $\mathcal{H}_\Pi$ representing its intended partner P_Π. The conversation transcript τ can be extracted from the messages sent and received in this phase. This phase runs for time T. At the end of the phase, if $< \mathcal{H}_i, \mathcal{H}_\Pi^* > (T) =$ `accept` then Λ_{auth} is set to `auth`, and is set to `reject`otherwise. In this phase the adversary interacts with oracles using the following query:

- Send$^{\text{auth}}(\pi_i^s, m)$: The adversary can use this query to send message m to oracle π_i^s. The oracle will respond according to the protocol specification, depending on its internal state, except that it replies with an error symbol $\perp$ if oracle π_i^s has state $\Lambda_{auth} =$ '`auth`'. Messages exchanged in this phase can be extracted into a Turing Test transcript according to the protocol transcript. (Send-queries in an `auth-state` are handled by the Decrypt-query below).

5.2.2.2.3 Post-accept phase Moreover, for the post-accept phase each oracle π_i^s keeps additional variables $(u_i^s, v_i^s, C_i^s, \theta_i^s, TS_i^s)$. Variables u_i^s, v_i^s are counters, which are initialized to $(u_i^s, v_i^s) := (0, 0)$. To simplify our notion we additionally define $u_0^0 := (0, 0)$. Variable C_i^s is a list of ciphertexts, which initially is empty. We write $C_i^s[u]$ to denote the u-th entry of C_i^s. Variable θ_i^s stores a pair of indices $\theta_i^s \in [\ell] \cup \{0\} \times [d] \cup \{0\}$. If oracle π_i^s accepts having a matching conversation to some other oracle π_j^t, then θ_i^s is set to $\theta_i^s := (j, t)$. Otherwise it is set to $\theta_i^s := (0, 0)$. TS_i^s is a monotonically increasing list of timestamps that is updated with each Decrypt-query. We write $TS[u]$ to denote the u-th entry of TS_i^s and $TS_i^s.prev$ to denote the last item of the list.

We will furthermore assume that the key k consists of two different keys $k = (k_{\mathsf{enc}}^{\rho}, k_{\mathsf{dec}}^{\rho})$ for encryption and decryption. Their order depends on the role $\rho \in \{\mathsf{Initiator}, \mathsf{Responder}\}$ of oracle π_i^s.

An adversary may interact with these oracles by issuing the following queries:

- $\mathsf{Encrypt}(\pi_i^s, m_0, m_1, \mathsf{len}, H)$: This query takes two messages m_0 and m_1, length parameter len, and header data H. If $\Lambda \neq$ '`accept`' then π_i^s returns $\perp$.

  Otherwise, it proceeds as depicted in figure 5-3, depending on the random bit $b_i^s \xleftarrow{\$} \{0, 1\}$ sampled by π_i^s at the beginning of the game and the internal state variables of π_i^s.

- $\mathsf{Decrypt}(\pi_i^s, C, H, ts)$: This query takes as input a ciphertext C, header data H, and a message timestamp t. If π_i^s has $\Lambda \neq$ '`accept`' then π_i^s returns $\perp$. Otherwise it proceeds as depicted in figure 5-3. In addition to the values returned described in the figure, during a Decrypt-query the oracle will also respond according to its protocol specification and update its state variables. This includes the possibility of moving to a `reject-state`and aborting the protocol.

Note. A Send-query refers to $\mathsf{Send}^{\mathsf{pre}}$ if the oracle is in the Pre-accept phase and Decrypt-query if it is in the Post-accept phase.

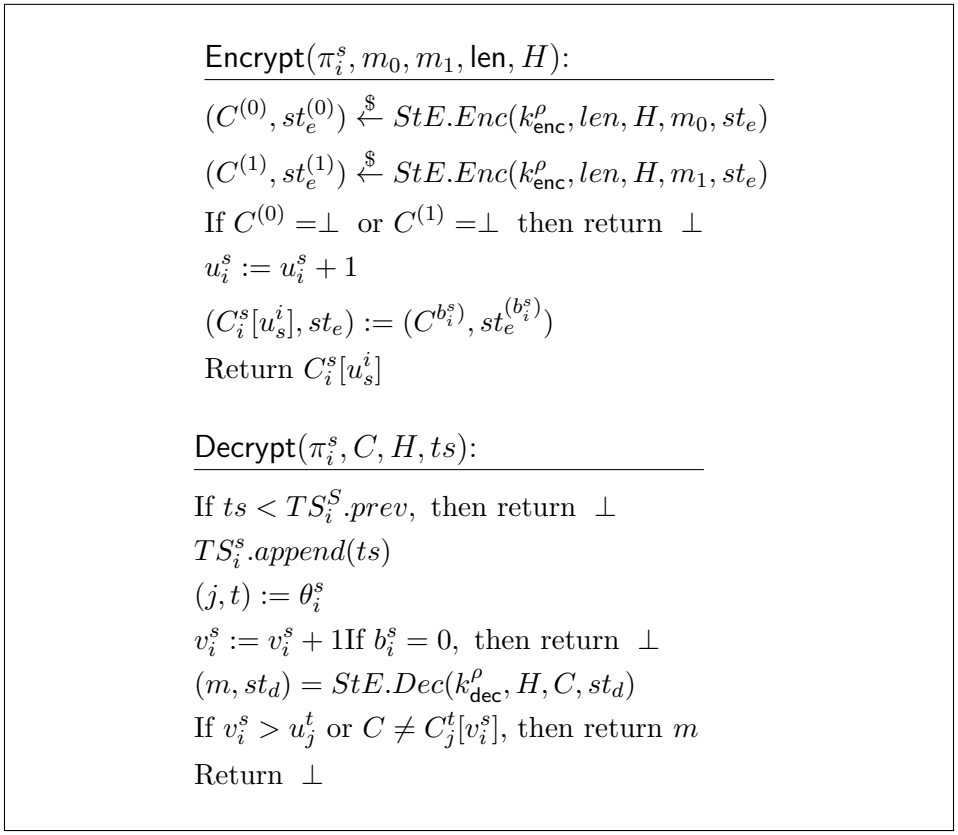

Figure 5-3. Encrypt and Decrypt oracles in the HASC security experiment

The Send-query enables the adversary to initiate and run an arbitrary number of protocol instances, sequentially, and provides full control over the communication between all parties. The Reveal-query may be used to learn the session keys used in previous protocol executions. The Corrupt-query allows the attacker to learn sk_i of party P_i, it may for instance be used by $\mathcal{A}$ to impersonate P_i.

5.2.2.3 Security Defintion

We borrow from [2] the notion of *matching conversations* in order to define correctness and security of a HASC protocol precisely.

We denote with $T_{i,s}$ the sequence that consists of all messages sent and received by π_i^s in chronological order (not including the initialization-symbol $\top$). We also say that $T_{i,s}$ is the *transcript* of π_i^s. For two transcripts $T_{i,s}$ and $T_{j,t}$, we say that $T_{i,s}$ is a prefix of $T_{j,t}$, if $T_{i,s}$ contains at least one message, and the messages in $T_{i,s}$ are identical to and in the same order as the first $|T_{i,s}|$ messages of $T_{j,t}$.

Definition 5.2 (Matching Conversations)**.** *We can say π_i^s has a* matching conversation *to π_j^t, if*

- *$T_{j,t}$ is a prefix of $T_{i,s}$ and π_i^s has sent the last message(s), or*
- *$T_{i,s} = T_{j,t}$ and π_j^t has sent the last message(s).*

The security of a HASC protocol is defined by requiring that either all data is transmitted over an authenticated and confidential channel or parties will be able to recognize if this is not the case.

This notion is captured by a game, played between an adversary $\mathcal{A}$ and a Challenger $\mathcal{C}$. The challenger implements the collection of oracles $\{\pi_i^s : i \in [\ell], s \in [d]\}$. At the beginning of the game, the challenger generates ℓ long-term key pairs (pk_i, sk_i) for all $i \in [\ell]$. The adversary receives the public keys $pk_1, ..., pk_\ell$ as input. Now the adversary

may start issuing $\mathsf{Send}^{\mathsf{pre}}$,$\mathsf{Send}^{\mathsf{auth}}$,Reveal, Corrupt, Encrypt, Decrypt queries. Finally, the adversary outputs a triple (i, s, b') and terminates.

Definition 5.3. *Assume a "benign" adversary $\mathcal{A}$, which picks two arbitrary oracles π_i^s and π_j^t and performs a sequence of* Send*-queries by faithfully forwarding all the messages between π_i^s and π_j^t for the Pre-accept, Auth, and Post-accept phases. Let $k_i^s = (k_{\mathsf{enc}}^{\mathsf{Initiator}}, k_{\mathsf{dec}}^{\mathsf{Initiator}})$ denote the key computed by π_i^s and let $k_i^s = (k_{\mathsf{enc}}^{\mathsf{Responder}}, k_{\mathsf{dec}}^{\mathsf{Responder}})$ denote the key computed by π_j^t. We say that a HASC protocol is* correct*, if for this benign adversary and any oracles π_i^s and π_j^t it always holds that*

1. *Both oracles have $\Lambda_{pre} = \texttt{accept}$.*
2. *Both oracles have $\Lambda_{auth} = \texttt{auth}$.*
3. *$k_i^s = k_j^t \in \mathcal{K}$*

Furthermore, we require that for all messages $m \in \{0,1\}^*$, lengths field $\mathsf{len} \in \mathbb{N}$ that are valid for m, roles $\rho \in$

{Initiator, Responder}, keys $k = (k_{\mathsf{enc}}^{\rho}, k_{\mathsf{dec}}^{\rho})$, headers $H \in \{0,1\}^*$, and encryption/decryption states $st_e, st_d \in \{0,1\}^*$ holds that $StE.Dec(k_{\mathsf{dec}}^{\rho}, H, StE.Enc(k_{\mathsf{enc}}^{\rho}, \mathsf{len}, H, m, st_e), st_d) = m$.

Definition 5.4. *We say an adversary $(t, \epsilon)-$breaks an HASC protocol, if $\mathcal{A}$ runs in time t, and at least one of the following two conditions holds:*

1. *When $\mathcal{A}$ terminates, then with probability at least ϵ there exists oracles π_i^s, π_j^t such that:*
   - *π_i^s, π_j^t 'accept' ($\Lambda_{pre} = \texttt{accept}$) with each other as their intended partner.*
   - *π_i^s, π_j^t do not have a matching conversation with each other at the end of the Accept phase.*
   - *Either π_i^s or π_j^t have $\Lambda_{pre} = \texttt{auth}$ at the end of the Auth phase.*

If π_i^s or π_j^t authenticate in the above sense, we say that the oracle authenticates maliciously.

2. *when $\mathcal{A}$ terminates and outputs a triple (i, s, b') the following conditions holds:*

   - *There exists π_j^t such that π_i^s has a matching conversation with π_j^t, π_i^s and π_j^t are both in* `accept-state` *at time t*

   Then the probability that b' equals b_i^s is bounded by

$$|Pr[b_i^s = b'] - 1/2| \geq \epsilon$$

   . If an adversary $\mathcal{A}$ outputs (i, s, b') such that $b' = b_i^s$ and the above conditions are met, then we say that $\mathcal{A}$ answers the encryption-challenge correctly.

We say a protocol is $(t, \epsilon) - secure$ if it is correct and there exists no adversary that $(t, \epsilon) - breaks$ it.

5.2.2.4 Relation to ACCE security definition

Note that an HASC protocol can be constructed in a three-step approach.

1. (CKA part) First an unauthenticated contributory key agreement (CKA) protocol is executed. This protocol makes no guarantees beyond providing an unpredictable session key so long as one honest party was involved.

2. (Authentication part) The session key is then used in a protocol's a posteriori (Human) authentication stage. This stage may overlap with the beginning of the protocol. This stage guarantees the authenticity of the communication partner.

3. (Symmetric part) At this point a new session key can be derived or the session key can be continued to be used in a symmetric encryption scheme providing integrity and confidentiality.

This approach maintains the original modularity of the ACCE definition, with the change of moving authentication to an additional phase that occurs after session key establishment.

Similar to ACCE, we argue that since the purpose of these protocols is to establish an authenticated confidential channel, indistinguishability and non-malleability of encrypted messages rather than full indistinguishability of session keys is sufficient.

5.3 LATENT Protocol

We now describe the LATENT protocol, our construction for a secure HASC scheme.

5.3.1 Network Model

The network model we assume is an unreliable channel such as UDP. While our protocols can certainly be implemented in combination with some form of packet retransmission, use of TCP specifically (using a standard TCP stack) will induce confounding effects due to the fact that applications do not receive fine-grained data on packet latency. This data is critical to implementing a latency-based security protocol. While, it is possible for channels with reliable delivery to provide this data (e.g. through TCP with kernel instrumentation or higher-level reliable protocol such as QUIC [56]), this would move us away from how video communications happen in practice. Additionally, we assume that the adversary has the ability to interfere with any existing reliability mechanisms in the network.

5.3.2 Design

The core contribution of our paper is our protocol for *a posteriori* authenticated key agreement. This formalizes the ideas laid out in prior MitM detection and forced latency works [22, 81, 88, 101]. In this section we will describe the details of our protocol, Latency-based Authentication Through Encrypted Network Telecommunications

(LATENT).

Acknowledgements. LATENT aims to address the technical problems that existing forced latency protocols face. First and foremost, in order to ensure that a party is aware of the receipt of its messages and the time of receipt we introduce acknowledgements. These acknowledgements are embedded in future committed packets so they are opaque to adversaries and cannot be tampered with. We make use of two acknowledgements per packet sent, one that is sent back immediately upon receipt to allow for measuring that baseline latency for sending and receiving packets (type 1). The second acknowledgement does not get sent back until its associated packet has been opened (type 2). This allows for measuring the actual latency for the communication with the protocol's artificial latency factored in.

The acknowledgements are in the form of random challenge strings. The type 1 challenge is sent in the clear so that the receiving party can immediately send it back. Due to the use of an AEAD, if the challenge was modified the recipient would detect it. While an MitM could read this challenge in transit, that poses little risk. It could send the acknowledgement of the challenge back earlier than the receiving party, but that would simply make the measured baseline latency appear lower. This would only make it easier to detect the increased latency caused by the MitM's presence. Since all acknowledgements are sent back within commitments, the MitM is unable to tamper with these in transit surreptitiously.

Human Authentication. The general intuition for LATENT comes from the idea that human's latent ability to authenticate each other operates as shared secret when no other pre-shared secret exists. Since no one aside from the honest communicating parties should be able to produce the transcript of an authenticating conversation, the messages involved in this conversation are unknown and unpredictable to an eavesdropper. Thus, an adversary should not be able to produce sensible responses until it sees these messages. As we will see in section 5.4, LATENT is designed so

that the security of the scheme reduces to the hardness of the Turing Test. Thus, successfully forging a significant amount of messages in LATENT would imply that an adversary is able to reliably win the Turing Test defined in the previous section, which is assumed to be a hard problem.

Real-time Synchronicity. LATENT is also designed so that each party can perform the authentication procedure simultaneously. This allows for both the ability for parties to independently assess whether a MitM is present as well as send and receive messages at the same time. This reduces the need of both parties staying in sync and also opens up a broader set of choices for the communication protocol underlying LATENT such as real-time audio or video.

Amount of Delay. LATENT additionally requires that the amount of forced latency be a fixed parameter of the protocol. The security of a forced latency protocol relies on detecting a substantial increase in latency if a MitM is present. However, given the lack of prior shared secrets it does not seem possible to allow this value to be various without giving the MitM the ability to influence this parameter. If the latency parameter is compromised, the MitM can adjust the value to hide its presence in all cases.

5.3.3 Overview

The high level operation of the protocol is as follows. While there is a pre-accept phase where a session key is established, we will focus on what happens during the auth and post-accept phases as that is where the core operation of LATENT occurs. The protocol involves two parties asynchronously sending packets to each other. Unlike prior work, this enables for simultaneous communication of the two parties. For a given packet a party wishes to send, it first sends a commitment to the message in the form of a ciphertext of the packet under a key it produces ephemerally as a function of the session key. After a specified delay (we model this delay as a specific

number of packets with the assumption that packets are being sent out from a party at regular intervals[2]), it sends another packet that contains an opening that allows the packet to be read. It will also obtain at some point an encrypted acknowledgement that the other party received this packet and an opening that allows it to view this acknowledgment. This allows the party to compute the channel latency and check whether it matches the expected delay.

Our claim, which we will later prove, is that any active adversary on this connection will be forced to significantly increase the latency of packets in order to hide its presence.

5.3.4 Building Blocks

In order to build LATENT, we assume the following cryptographic functionalities:

- DH : A Diffie-Hellman contributory[3] key exchange protocol. At the end of the protocol each party will arrive at a key.
- $\mathsf{KDF}(K, n) \rightarrow k$: A Key Derivation Algorithm that takes in a long term key K and a nonce n and returns a key k. We assume this KDF meets the definition of a PRF as well.
- An AEAD scheme consisting of the tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ that are defined as follows:
  - $\mathsf{KeyGen}(1^n, K) \rightarrow k, n$: Takes in as input 1^n where n is the security parameter and a long term key K. Then it samples $n \xleftarrow{\$} \{0,1\}^\ell$, where ℓ is the length of nonces for this AEAD scheme. It returns $k := \mathsf{KDF}(K, n)$ and n.
  - $\mathsf{Enc}(k, M, D) \rightarrow C$: Takes as input a key k, a plaintext M, additional data to authenticate D and returns a ciphertext C.

[2]We can constrain this as part of the system operation

[3]Contributory key exchange requires that neither party can force the derived key to be a particular value.

– $\mathsf{Dec}(k, C, D) \rightarrow M \in \{\perp\} \cup \{0,1\}^*$: Takes in a key k, a ciphertext C, and additional data to authenticate D and if the ciphertext was validly generated under k it returns the plaintext. Otherwise it returns $\perp$.

We assume this AEAD is secure in the definition of Rogaway [83].

5.3.5 Construction

The forced latency features and asynchronous nature of LATENT make the protocol fairly complex with many important pieces to discuss. However, at its core, the protocol consists of two parties that negotiate a key and then exchange packets within an encrypted channel. The number of packets to delay by, δ, is a fixed system parameter.

Pre-accept phase. In this phase a DH is executed between the parties and each party P arrives at a session key K_P. If the parties are honest then this key will be the same. (i.e. $\Lambda =$ '`accept`').

Auth phase. In this phase, each party will execute the LATENT protocol for each packet they wish to send. All data sent over this phase is encrypted with the parties session key K_P. Based on the LATENT protocol's detection of an MitM as well as the contents of the packets exchanged, each party will decide on an `auth-state`. This phase ends when both communication partners reach an `auth-state` (i.e. $\Lambda_{auth} =$ '`accept`').

Post-accept phase. This phase is entered, when both communication partners reach an `auth-state`. In this phase, each party will continue to execute the LATENT protocol for each packet they wish to send; however, at this point the channel is assumed to be authenicated and secure. All data sent over this phase is encrypted with the parties session key K_P.

During the Auth and Post-accept phase, each party P runs the procedures Send and

Receive. This procedures will share state, but are guaranteed to only run sequentially.

- $\mathsf{Send}(m_i, t) \to L_i^\rho$: Takes in a message m_i and a timestamp for this message t and returns a LATENT packet L_i^ρ. This method encrypts the message under a one-time packet key and generates a pair of challenges for determining baseline (type 1) and overall latency (type 2). It packs the messages and challenges alongside acknowledgments for the most recent challenges received and the nonce to compute the $(i-\delta)$th packet key into the returned LATENT packet L_i^ρ. The time the message is sent is also recorded so that latency can be computed.

- $\mathsf{Receive}(L_j^{\bar{\rho}}, t) \to m_{j-\delta}$: Takes in a LATENT packet $L_j^{\bar{\rho}}$ and a timestamp t and returns a message $m_{j-\delta}$. This message processes a received LATENT packet by using the included nonce to decrypt the $(j-\delta)$th packet that was received. The included challenges are stored so they can be used to acknowledge the received packet. The time t of packets being acknowledged is recorded so baseline and overall latency can be recorded.

The full details of these methods are included in figure 5-4.

The communicating parties call these methods as they exchange packets. As they do so, the following state variables are maintained:

- i: the current packet number. Starts at 0 and is incremented with each new packet to send.

- $\vec{N}$: A list of packet nonces. These will be used to commit to and open packets. We say that for $i < 0, N_i$ is the i-th element of $\vec{N}$.

- $\vec{A}$: A list packet challenge pairs. $A_{i,\tau}$ denotes challenge type $\tau \in \{1, 2\}$ for the i-th packet sent.

$\mathsf{Send}(m_i, t)$:

$k_i^\rho, n_i \leftarrow \mathsf{KeyGen}(1^n, K)$

$a_{i,1}^\rho, a_{i,2}^\rho \xleftarrow{\$} \{0,1\}^w$

$A_{i,1} := a_{i,1}^\rho$

$A_{i,2} := a_{i,2}^\rho$

$\mathsf{ACK}_1, a_{\mathsf{ACK}_1,1}^{\bar{\rho}} := \vec{\mathsf{ACK}}_1.pop()$

$\mathsf{ACK}_2, a_{\mathsf{ACK}_2,2}^{\bar{\rho}} := \vec{\mathsf{ACK}}_2.pop()$

$n_{i-\delta}^\rho := N_{i-\delta}$

$\mathsf{M} = a_{\mathsf{ACK}_1,1}^{\bar{\rho}} \| \mathsf{ACK}_1 \| m_i \| a_{i,2}^\rho \| a_{\mathsf{ACK}_2,2}^{\bar{\rho}} \| \mathsf{ACK}_2$

$\mathsf{D} = i \| a_{i,1}^\rho \| n_{i-\delta}$

$\mathsf{C} := \mathsf{Enc}(k_i^\rho, \mathsf{M}, \mathsf{D})$

$L_i^\rho = \mathsf{C} \| \mathsf{D}$

$N_i := n_i^\rho$

$i := i + 1$

$T_{i,1} := t$

Return L_i^ρ

$\mathsf{Receive}(L_j^{\bar{\rho}}, t)$:

$\mathsf{C}, \mathsf{D} = L_j^{\bar{\rho}}$

$j, a_{j,1}^{\bar{\rho}}, n_{j-\delta}^{\bar{\rho}} := \mathsf{D}$

$\vec{\mathsf{ACK}}_1.push((j, a_{j,1}^{\bar{\rho}}))$

$C_j := (j, \mathsf{C}, \mathsf{D})$

If $j - \delta >= 0$ AND $n_{j-\delta}^{\bar{\rho}} \neq \perp$

  $k_{j-\delta}^{\bar{\rho}} \leftarrow \mathsf{KDF}(K, n_{j-\delta})$

  $(j - \delta, \mathsf{C}, \mathsf{D}) := C_{j-\delta}$

  $\mathsf{M} := \mathsf{Dec}(k_{j-\delta}^{\bar{\rho}}, \mathsf{C}, \mathsf{D})$

  If $\mathsf{M} = \perp$ then return $\perp$

  $a_{x,1}^\rho, x, m_{j-\delta}, a_{j-\delta,2}^{\bar{\rho}}, a_{y,2}^\rho, y := \mathsf{M}$

  $\vec{\mathsf{ACK}}_2.push((j - \delta, a_{j-\delta,2}^{\bar{\rho}}))$

  If $A_{x,1} = a_{x,1}^\rho$ then $T_{x,2} := t$

  If $A_{y,2} = a_{y,2}^\rho$ then $T_{y,3} := t$

  Return $m_{j-\delta}$

Return $\perp$

Figure 5-4. Methods for sending and receiving packets

- $\vec{T}$: A list of timestamps recording when a packet is sent and acknowledgement is received. $T_{i,z}$ denotes timestamps for the i-th packet sent. $z \in \{1,2,3\}$ where $z = 1$ is the time packet is sent, $z = 2$ is the time first acknowledgement is received, $z = 3$ is the time the second acknowledgement is received.
- $\mathsf{A}\vec{\mathsf{C}}\mathsf{K}_\tau$: A stack of challenges received of type τ. $\mathsf{A}\vec{\mathsf{C}}\mathsf{K}_\tau.pop()$ returns the most recent challenge received. $\mathsf{A}\vec{\mathsf{C}}\mathsf{K}_\tau.push(j, a^{\bar{\rho}}_{j,\tau})$ updates the most recent challenge received.
- $\vec{C}$: A list of received encrypted packets.

Additionally, we define the variables discussed in the methods as follows:

- m_i: the i-th data to send over.
- $\bar{\rho}$: Whichever party ρ is not.
- ACK_τ: The packet number of the most recent challenge of type $\tau \in \{1,2\}$ received from $\bar{\rho}$, but not yet acknowledged by ρ. If there is no challenge waiting this value is $\perp$.
- $a^{\rho}_{i,\tau}$: A challenge generated by ρ, of type $\tau \in \{1,2\}$ for packet number i. If $i = \perp$ then $a^{\rho}_{i,\tau} = \perp$
- $n^{\rho}_{i-\delta}$: the nonce used in generation of the one-time packet key by ρ for packet number $i - \delta$, where δ is the packet delay parameter. If $i - \delta < 0$, then the value of these field is $\perp$.

For succinctness we say that the tuple $\{\rho, i, \mathsf{ACK}_1, \mathsf{ACK}_2\}$ refers to the packet with the structure:

$$\mathsf{C}\|\mathsf{D}, \mathsf{C} := \mathsf{Enc}(k^{\rho}_i, \mathsf{M}, \mathsf{D})$$

$$\mathsf{M} = a^{\bar{\rho}}_{\mathsf{ACK}_1,1}\|\mathsf{ACK}_1\|m_i\|a^{\rho}_{i,2}\|a^{\bar{\rho}}_{\mathsf{ACK}_2,2}\|\mathsf{ACK}_2$$

$$\mathsf{D} = i\|a^{\rho}_{i,1}\|n_{i-\delta}$$

5.3.6 Illustrated Protocol Flow

Figure 5-5 illustrates how LATENT works for a single packet from Alice to Bob. WLOG, Bob uses the same process to authenticate its packets. The full LATENT protocol works by Alice and Bob following this protocol for all the packets they send. To make this protocol more comprehensible, we will break down what each message sent serves (Note: Variables not mentioned are not relevant for the authentication of m_i , but will serve a role in the authentication of other packets):

- **Message one**: serves to provide Bob with ciphertext committing to m_i.
- **Message two**: Bob sends ciphertext commitment with the first acknowledgement of m_i.
- **Message three**: Alice reveals the opening to the commitment to m_i. This allows Bob to compute k_i, and verify that the ciphertext and additional data, challenge A_i, and packet number i, have not been tampered with. At this stage Bob can read m_i and the second challenge $a^{\rho}_{i,2}$.
- **Message four**: Bob reveals opening to commitment containing the first acknowledgement. Alice can now verify Bob sent the first acknowledgement and use the time of receipt to compute the baseline RTT of the connection.
- **Message five**: Bob sends ciphertext commitment with the second acknowledgement of m_i.
- **Message six**: Bob reveals opening to commitment containing the second acknowledgement. Alice can now verify Bob sent the second acknowledgement and use the time of receipt to compute the forced latency RTT.

Note: Message two may occur anytime after message one. Message three occurs δ packets after message one. Message four occurs δ packets after message two. Message

five may occur anytime after message three. Message six occurs δ packets after message five.

5.3.7 Potential Extensions

Although it LATENT adds additional latency, this protocol should allow for viable communication where eavesdroppers can be detected. However, it may be preferable to remove artificial latency once both parties are assured there is no MitM present. As mentioned earlier, the latency parameter itself cannot be adjusted dynamically without significant security risks. Alternatively, once both parties are satisfied there is no MitM present, there could be an option to perform a Key Agreement protocol such as Diffie-Hellman Key Exchange. Thus, the parties would now have a shared secret they could use to communicate on a new channel without need for LATENT or fear of an MitM. In this way, LATENT may be used to bootstrap further security protocols.

5.4 Proving LATENT is a Secure HASC protocol

Theorem 5.1. *Let n be the security parameter, and ℓ be the nonce length, and w be the length of packet challenges. Assume that the AEAD scheme is secure, the* KDF *is secure, the* CDH *is hard. Furthermore assume that the Turing test is hard. Additionally suppose that the number of packets t sent by honest parties in this protocol will always be significantly larger than ϵ, the fabrication parameter in the Turing test. Then LATENT is a $(t', \epsilon) - secure$ HASC protocol for all $t' > \epsilon$.*

We will first provide a sketch of the proof to provide intuition and then detail the full proof.

Proof sketch.

Fundamentally, the objective is demonstrating that the adversary cannot win the authentication challenge if the session keys of two participants do not match. Clearly,

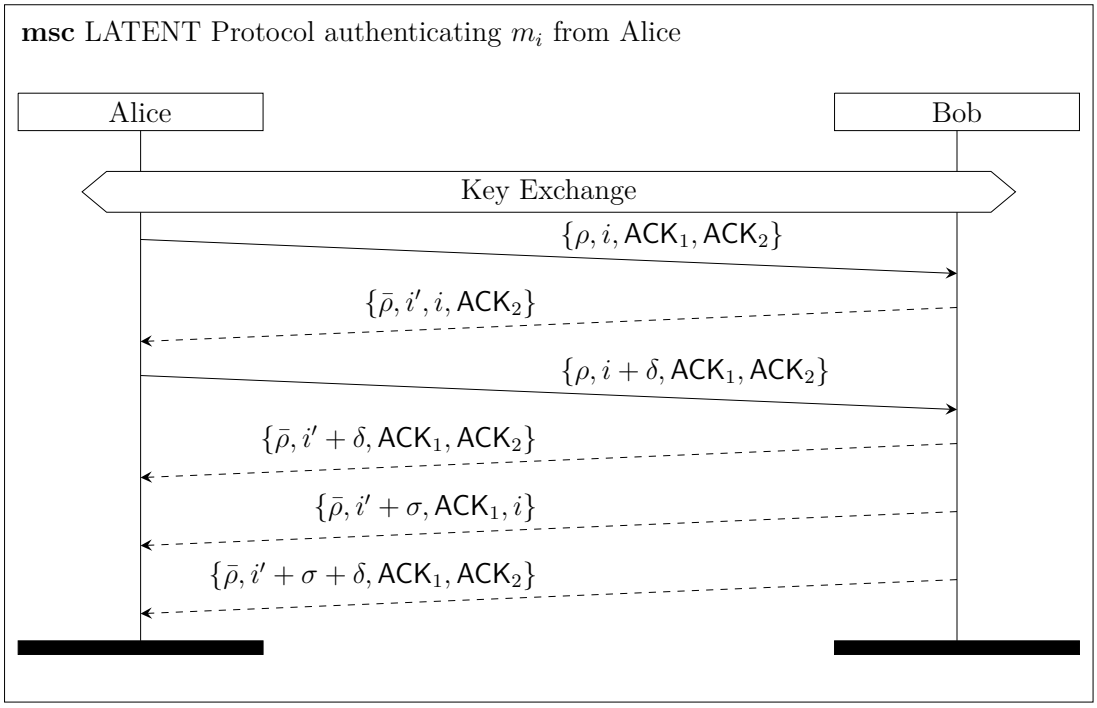

Figure 5-5. Illustration of how LATENT authenticates packets. Explanation of simplified packet structure: ρ denotes a message by Alice, while $\bar{\rho}$ denotes a message by Bob. The second field denotes the packet number. The third and fourth fields denote the packet receiving its first and second acknowledgement (Here ACK_1 and ACK_2 are placeholders if those acknowledgements are not relevant to the authentication of packet i).

if both parties are entering the post-accept phase with honestly generated keys, it is easy to see that the adversary should not have an advantage to winning the encryption challenge.

Thus the core of our proof is showing that our protocol ensures that both participants reject at the end of the auth phase if their keys do not match.

The fundemental idea we leverage is the very fact that the keys do not match and how our protocol mixes that key information into its messages. The adversary can never send over messages unmodified as every message encodes that parties key information and would reveal a mismatch. Thus, the adversary is forced into an active role and our goal is to demonstrate any active tampering with the protocol will be detectable.

The way our protocol proceeds is by forcing the adversary to make a detectable choice for each packet sent: either wait for the packet opening and then replace all the key dependent information with that of the recipient or attempt to forge a message.

Clearly, waiting for the packet openings has a straightforward detection process. In each packet, in addition to the message content is also acknowledgements of received packets. To delay these packets to fix them up is also to delay the acknowledgement of packets and thus parties can easily see this discrepancy in their measured latency.

Alternatively, an adversary can choose to just forge the messages sent and therefore not need to wait for packet openings. However, this is where the Turing Authentication game comes in. We allow for the adversary to be able to forge a small number of packets. However, for the adversary to forge enough packets to make it through the auth phase without also triggering the latency detection would require more fabricated packets than is considered possible in the Turing Authentication game. Thus, if an adversary was able to forge this many packets, it could also be used as a winning adversary for that game as well.

Thus, whether the adversary attempts to conceal their presence through waiting for packet openings and replacing the contents or by forging messages, they will not be able to avoid detection with non-negligible probability. And thus, the rest of the proof follows from this point.

□

5.4.1 Full Proof of Security

In order to prove Theorem 5.1 we will divide the set of all adversaries into two categories:

1. Adversaries that succeed in making an oracle authenticate maliciously. We call such an adversary an authentication adversary.

2. Adversaries who do not succeed in making an oracle auth maliciously, but which answer the encryption challenge. We call such an adversary an encryption-adversary.

Lemma 5.1. *The probability that an adversary can cause an oracle to authenticate maliciously is negligible.*

Proof. The proof proceeds in a sequence of games, where the first game is the real security experiment. We then describe several intermediate games that modify the original game step by step and argue or complexity assumptions imply each game is indistinguishable from the previous. We end up in the final game, where no adversary can break the security of the protocol.

Game 0. In this hybrid, $\mathcal{A}$ interacts as described for the HASC security experiment described in Section 5.2.2.

Game 1 (**Abort if two oracles sample the same key share**) This hybrid modifies the previous as follows:

In the event that any two oracles select the same DH key share abort. Since each oracle samples their own key share independently, we know that there is a negligible chance that two shares collide. Thus the adversary has a negligible advantage in distinguishing this game from **Game 0**.

Game 2 (**Guess which oracles will authenticate maliciously first, abort if wrong**) This hybrid modifies the previous as follows:

We as the challenger guess the indices of the oracles that will first authenticate maliciously. If some other oracles authenticate first, we abort. Since there are only a polynomial number of communicating pairs, this only modifies the win probability of the adversary by a polynomial factor. Thus the adversary still has a negligible advantage in distinguishing this game from **Game 1**.

Game 3 (**Abort on matching maliciously produced session keys**) This hybrid modifies the previous as follows:

In the event that two communicating parties without matching conversations during the pre-accept phase arrive at the same key, then abort. We know that honest parties with matching conversations will always arrive at the same key. By the properties of cyclic groups, for any x, y s.t. $g^x = g^y, x = y$. Thus, if the two parties arrive at keys g^{ar_1}, g^{br_2} respectively, the adversary must have picked r_1, r_2 s.t. $ar_1 = br_2$, which would only happen with negligible chance. Thus the adversary has a negligible advantage in distinguishing this game from **Game 2**.

Game 4 (**If $\mathcal{A}$ is not party to a connection, send random data**) This hybrid modifies the previous as follows:

If $\mathcal{A}$ is not party to a connection, and thus the communicating parties having

matching conversations at the end of the pre-accept phase and arrive at the same session key, then $\mathcal{C}$ has the communicating oracles send encryptions of random data instead of following the LATENT protocol.

If communicating parties succeed in arriving at an honestly generated key, there are only certain strategies for $\mathcal{A}$ to win. Dropping packets or mauling messages would trivially not give the adversary and advantage in distinguishing this game.

If the adversary knows the key, then it could distinguish this hybrid from the previous one by decrypting messages. However, if the key is honestly generated then for the adversary to learn the key knowing only the key shares would break the CDH assumption.

Alternatively, if the adversary does not know the key but still distinguishes this hybrid from the previous one with non-negligible advantage, we can use it to build a solver for the AE game. Assume a real experiment where all data across all honest connections is honestly generated and an ideal experiment where all data is random data. Now consider that the adversary can run any number of honest connections. We proceed by hybrid argument, where the 0-th hybrid is the real experiment and the final hybrid is the ideal experiment. Let the i-th hybrid be such that all connections up to the i-th connection produced by the adversary only have random data sent over, and all later ones do not. We argue that hybrid i and i+1 are indistinguishable, because if there existed an adversary $\mathcal{D}$ that could distinguish the two we could do the following: We can build $\mathcal{E}$, an AE solver with $\mathcal{D}$ as a subroutine. $\mathcal{E}$ runs $\mathcal{D}$ as well as the oracles that are part of the HASC game that $\mathcal{D}$ interacts with. Upon querying the challenger on the messages to encrypt for the i+1-th connection, it then sends over the received ciphertexts between the parties on the connection. $\mathcal{E}$ returns $\mathcal{D}$ output. Since $\mathcal{D}$ with non-negligible chance can tell whether the data being sent over the i+1-th connection is random or not, it will detect whether the encryption

oracle actually encrypted the queried messages or not. Since we assume the AE scheme is secure this is a contradiction, so there must be a negligible chance of distinguishing the hybrids.

Game 5 (**Abort if $\mathcal{A}$ mauls commitment**) This hybrid modifies the previous as follows:

If $\mathcal{A}$ mauls any ciphertext commitment sent, abort the protocol. By **Game 3** we know that this must happen before $\mathcal{A}$ is able to open the commitment and see the contents. However, we assumed that the ciphertext commitment was generated by a secure AEAD, which clearly satisfies the requirements for a nonmalleable commitment scheme. Thus, any attempt to maul would cause the receiving part to abort after attempting to decrypt. As a result, this would have a negligible effect on the adversaries win probability.

Game 6 (**Abort if $\mathcal{A}$ waits for packet opening**) This hybrid modifies the previous as follows:

Upon intercepting a packet, if $\mathcal{A}$ waits until it received the packet opening nonce before forwarding on the packet, then $\mathcal{C}$ aborts the protocol.

If either communicating party detects that packets have a 2δ delay, they will abort the protocol. Thus, $\mathcal{A}$ must avoid introducing this delay in order to win.

If $\mathcal{A}$ naively waits for the packet opening nonce before sending on a packet then that results in a δ delay. Since they have keep the receiving parties view of the protocol correct, they have to wait another δ packets before sending the packet opening nonce to the receiving party. Thus, the receiving party is able to read a packet 2δ packets after it was initially sent. Since this is also when the receiver is finally able to send the second acknowledgement of the packet, the sender will be able to detect this delay. Thus, the adversary cannot avoid the sender aborting in this scenario.

Clearly if $\mathcal{A}$ adds additional delay, that will only make the attack easier to detect so that strategy would trivially fail.

Thus, the adversary needs to somehow reduce the delay they introduce by waiting. Time cannot be modified in the real world, but the adversary can attempt to create an illusion of this by artificially modifying the latency of the network. The more the measured baseline RTT is increased, the smaller the additional latency appears. However, if $\mathcal{A}$ tries to hide their attack by adding a δ delay to baseline RTT measured, then removing this latency while performing their attack then there is still an issue. The adversary is still causing an additional δ delay to elapse. So the sender will still see that there is a 2δ delay in the network and abort as a result.

Thus, no matter what $\mathcal{A}$ attempts to do, the sender will still detect the additional delay. So there is no chance that $\mathcal{A}$ can win in this situation so aborting would have no impact at the probability of winning the HASC game.

Game 7 (**Abort if $\mathcal{A}$ can extract m_i from its commitment**) This hybrid modifies the previous as follows:

$\mathcal{C}$ aborts if the $\mathcal{A}$ is able to obtain m_i from its commitment, before receiving the commitment opening. There are two possibilities for this happening:

$\mathcal{A}$ can extract m_i if it is able to determine or influence the one-time packet key. While $\mathcal{A}$ knows the long term keys of each party, it has no knowledge of the random nonce used in key generation. Thus, in order to determine or influence the key, it would have to break the security of the KDF, which it can only do with negligible probability. Thus, aborting under this scenario would only negligibly affect the win probability.

$\mathcal{A}$ could also can extract m_i if it is able to break the AEAD scheme. However, we have assumed that this is only possible with negligible probability. So aborting

under this scenario would also only negligibly affect the win probability.

Game 8 (**Abort if $\mathcal{A}$ sends $> \epsilon$ fabricated messages**) This hybrid modifies the previous as follows:

$\mathcal{C}$ aborts if the $\mathcal{A}$ sends $\geq \epsilon$ fabricated messages. If the adversary is able to win in this scenario with non-negligible probability, then we can use it to win the Human Authentication game with non-negligible probability.

We construct the adversary of the Turing Authentication game as follows: It constructs the challenger for the HASC game using the Turing Authentication oracles as the communicating oracles in the HASC game. We have $\mathcal{A}$ as a subroutine in this adversary that interacts with this challenger. We know that with non-negligible probability, $\mathcal{A}$ will be able to send $> \epsilon$ fabricated messages without being detected. Thus, the Human oracle would not abort, so this adversary wins the Turing Authentication game. However, we assumed that winning Turing Authentication game only happens with negligible probability, so aborting in this scenario would only negligibly affect the win probability.

Game 9 (**Set oracle to be in `reject-state` after it receives ϵ messages**) This hybrid modifies the previous as follows:

Once one of the communicating oracles receives ϵ messages, it updates its internal state to be in the `reject-state`. We can see that is negligibly affects the win probability from the previous hybrid. To reach this scenario without the challenger aborting means that $\mathcal{A}$ sends $< \epsilon$ fabricated messages due to **Game 8**. Thus, at least one message must have its content unaltered.

By **Game 6**, $\mathcal{A}$ does not wait for a packet opening thus it could not have fixed up the message. So the difference in keys is exposed and $\mathcal{A}$'s interference is revealed and the oracles would reject.

Thus since we have shown that there is a negligible difference between **Game 0** and **Game 9**, the authentication challenge cannot be won with non-negligible probability.

□

Lemma 5.2. *The probability that an adversary can win the encryption challenge is negligible.*

Proof. he proof proceeds in a sequence of games, where the first game is the real security experiment. We then describe several intermediate games that modify the original game step by step and argue or complexity assumptions imply each game is indistinguishable from the previous. We end up in the final game, where no adversary can break the security of the protocol.

Game 0. In this hybrid, $\mathcal{A}$ interacts as described for the HASC security experiment described in Section 5.2.2.

Game 1 (**Abort if two oracles authenticate maliciously, and output random** b') This hybrid modifies the previous as follows:

In this game, if a pair of oracles authenticate maliciously we abort and output a random b' value. We saw from Lemma 1 that the probability of this happening is negligible. As a result, this would have a negligible effect on the adversaries win probability.

Game 2 (**If** $\mathcal{A}$ **is not party to a connection, send random data during the post-accept phase**) This hybrid modifies the previous as follows:

If $\mathcal{A}$ is not party to a connection, and thus the communicating parties having matching conversations at the end of the pre-accept phase and arrive at the

same session key, then $\mathcal{C}$ has the communicating oracles send encryptions of random data during the post-accept phase.

If communicating parties succeed in arriving at an honestly generated key, there are only certain strategies for $\mathcal{A}$ to win. Dropping packets or mauling messages would trivially not give the adversary and advantage in distinguishing this game.

If the adversary knows the key, then it could distinguish this hybrid from the previous one by decrypting messages. However, if the key is honestly generated then for the adversary to learn the key knowing only the key shares would break the CDH assumption. Additionally, we know from **Game 1** only oracles with honestly generated keys will enter the post-accept phase. As a result, this would have a negligible effect on the adversaries win probability.

Game 3 (**Alway output a random** b') This hybrid modifies the previous as follows:

In all cases, if no one has aborted by time t, the challenger ends the protocol and outputs a random b'. Since all messages being sent are random, the adversary has no additional advantage to winning the encryption challenge. Thus in this game, the challenger and adversaries output will be identically distributed.

Thus since we have shown that there is a negligible difference between **Game 0** and **Game 3**, the encryption challenge cannot be won with non-negligible probability.

□

Thus, with Lemma 1 and Lemma 2 we see that no adversary is able to win either the authentication or encryption challenge with non-negligible probability and thus LATENT is a $(t', \epsilon) - \textit{secure}$ HASC protocol for all $t' > \epsilon$.

5.5 Experimental Results

In order to assess the real world applicability of our protocol, we found it necessary to test whether it would behave as expected in practice. In order to do this we chose to build a video chat application that implements the LATENT protocol. Then we analyzed the recorded latencies to see whether the delay manifested as expected.

We chose to test LATENT on top of video chat as it demonstrates the simultaneous communication nature of LATENT. Additionally, we believe video chat is well suited to protocols that rely on Human Authentication such as ours. While much of the high entropy these protocols depend on comes from the content of the message, the audio and visual aspects of a video chat only serve to increase this property. Beyond message content, this factors in an individual's particular delivery style or facial expressions. Video chat also provides a constant stream of data for more precise latency measurements, as real-time text messaging will likely have additional intermessage delays that would add noise to the system.

5.5.1 Implementation

In order to build LATENT, we needed an operational video chat application that we could extend to support our protocol. We decided to extend webRTC [35] as it is an p2p open source platform that, while still under development, is being deployed in popular web chat applications. We feel by building on top of such a popular platform we further demonstrate that our system can offer practical usage.

We built our version of webRTC-LATENT by modifying the native c++ webRTC desktop application, configured over UDP, to implement the methods from figure 5-4. While it was possible to build our protocol on top on a reliable transport, as we mentioned earlier this was not a goal of the paper. WebRTC consists of a signaling phase where clients communicate with a server to establish a p2p session and a session

key is established. Then the clients exchange messages directly in a channel protected by DTLS, potentially with a TURN server relaying messages if clients are behind a NAT. The signaling phase and communicating phases map fairly clearly to the pre-accept and post accept phases in our definition.

We chose to look in particular at the scenario where a malicious signaling server has also compromised the TURN server used. In this situation, since messages have to travel through a relay anyways, the present of a MitM would not change the expected path of packets. The malicious server would ensure that the relay is able to decrypt any packets that it receives in the absence of LATENT. The MitM follows the strategy of delaying until it receives packet openings and then repackaging the messages under the new key. As we argued in 5.4, the adversary is limited in what it can get away with so ensuring our protocol works in this scenario should be sufficient to demonstrate practical security.

In Figure 5-8 we provide a visualization of how LATENT impacts calls in practice in the honest and MitM settings. As we can see, the presence of an MitM would add a noticeable additional delay to any communications.

5.5.2 Experimental Setup

We ran our experiments on Ubuntu 16.04 VM running with 4GB RAM each and 2.6 GHz Intel Core i7 CPU. Tests were run over the regular lab network. Key exchange was performed over DTLS and we used tweetnacl [3, 95] as our AEAD. We instantiated our protocol with $\delta = 10$ which translates into a delay of roughly 50ms[4].

Our experiments consisted of recording the timing data gathered by LATENT over a series of 10 video calls with a 5 minute duration each. Using this data, we can examine whether the latency manifests as expected as well as assess the effectiveness of detecting an MitM. These calls were also used to qualitatively verify that our video

[4]We heuristically found this delay to be easy to detect in normal circumstances

chat could still be used to viably communicate despite the added latency.

To see how the latency manifests, we will examine the density plots and means of the averaged data. To see the accuracy of our detection, we calculate given the average latency so far in ten second increments (to model performing this check every 10 seconds). Then if the observed latency appears to be greater than 2δ we predict an MitM is present. This detection strategy is dynamic and ongoing throughout the call. We compare the predictions to the actual scenario and average between our trials to get our final results. An important advantage of our protocol is that it is resilient to some degree of packet loss, since we compute average latency over many packets.

While this detection strategy was suitable in the evaluation of our proof of concept, we note that in practice a stricter detection would be preferable. Alternative approaches could have multiple measurement windows, where a majority vote after a specified number of windows would determine whether a MitM is present. A very lossy channel could be handled by using a rolling average to compute latency. A more aggressive strategy could see if any any measured latency is $2 * \delta$. As we see in the results section, this approach might be viable when parties are close-by or the network has a stable connection.

5.5.3 Results and Analysis

The results of our experiments are shown in Figures 5-6 and 5-7. From Figure 5-6 we see that we can reliably measure latency and that the MitM adds a clear difference to the latency distributions.

We can see that the overall measured latency adds about an additional 50ms delay on average with our setting of $\delta = 10$. In the scenario without an MitM, it seems that the additional delay is consistent with respect to the baseline measurements.

We note that the means seem to be skewed to the right somewhat, and this appears to be the result of rare but large spikes in the latency of the connection. However,

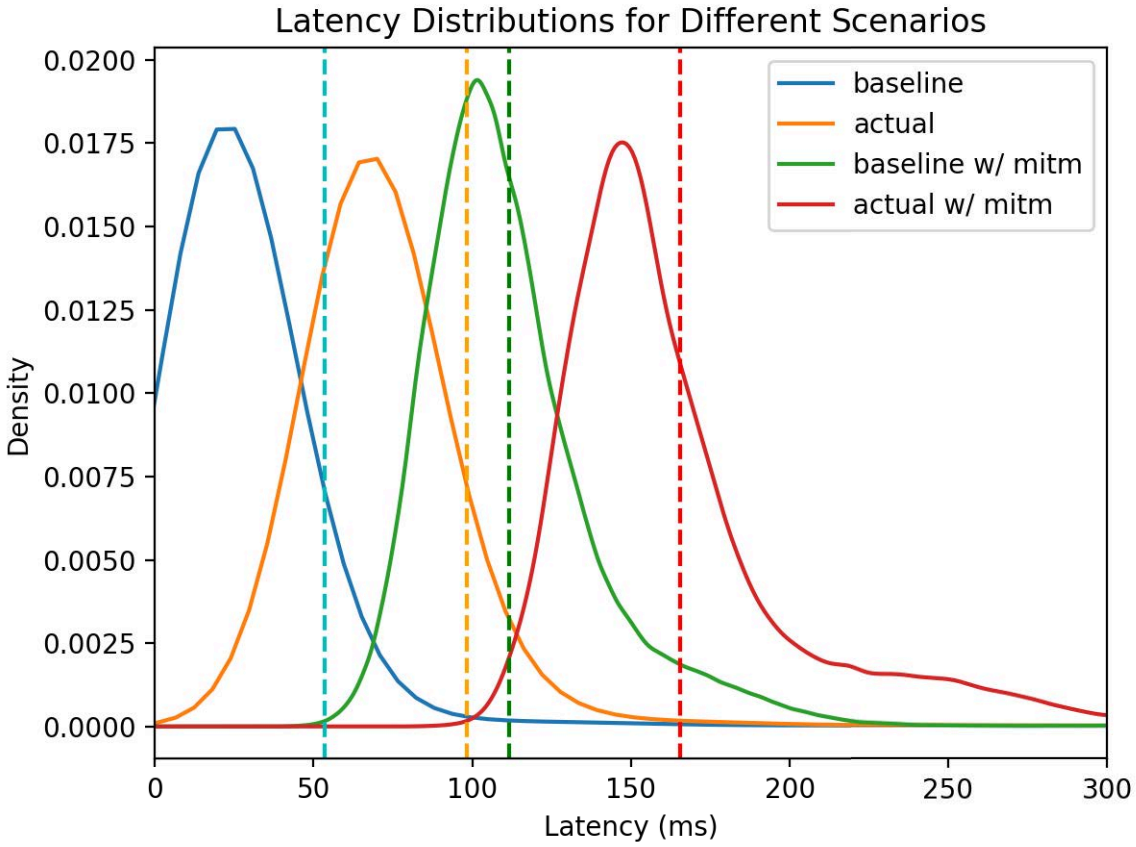

Figure 5-6. Comparison of latency distributions when MitM is present and absent

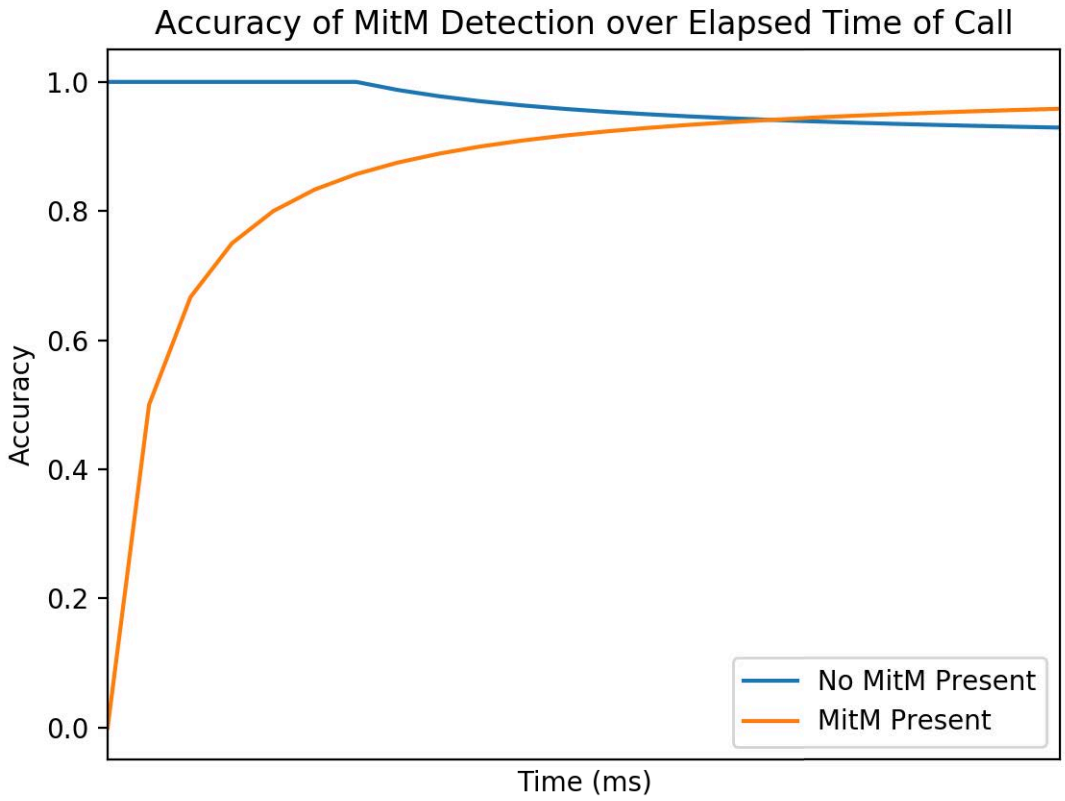

Figure 5-7. Average accuracy of MitM detection over the course of a video call

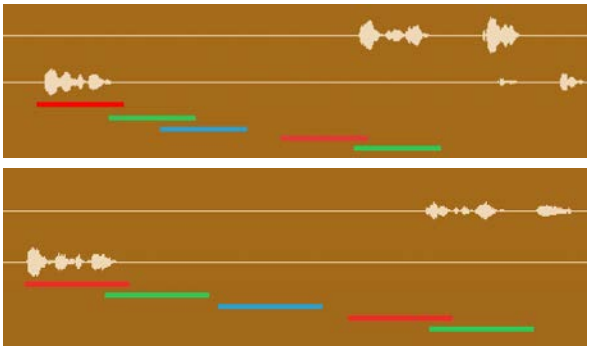

Figure 5-8. Visualization of the impact of LATENT on a call. In each diagram, we have an audio rendering of one party asking a question and the other party responding from the perspective of the first party. The top diagram has no MitM present, while the bottom diagram does. The bars under the audio represent the following 5 stages: 1) the encrypted packets being sent by the first party. 2) The openings of those encrypted packets. 3) The second party hearing the sent audio. 4) The first party receiving encrypted packets with the second party's response. 5) The first party receiving the packet openings and hearing the received audio. Red bars represent encrypted packets and green bard represent the opening of those packets. Blue bars represent when the audio is heard by the second party.

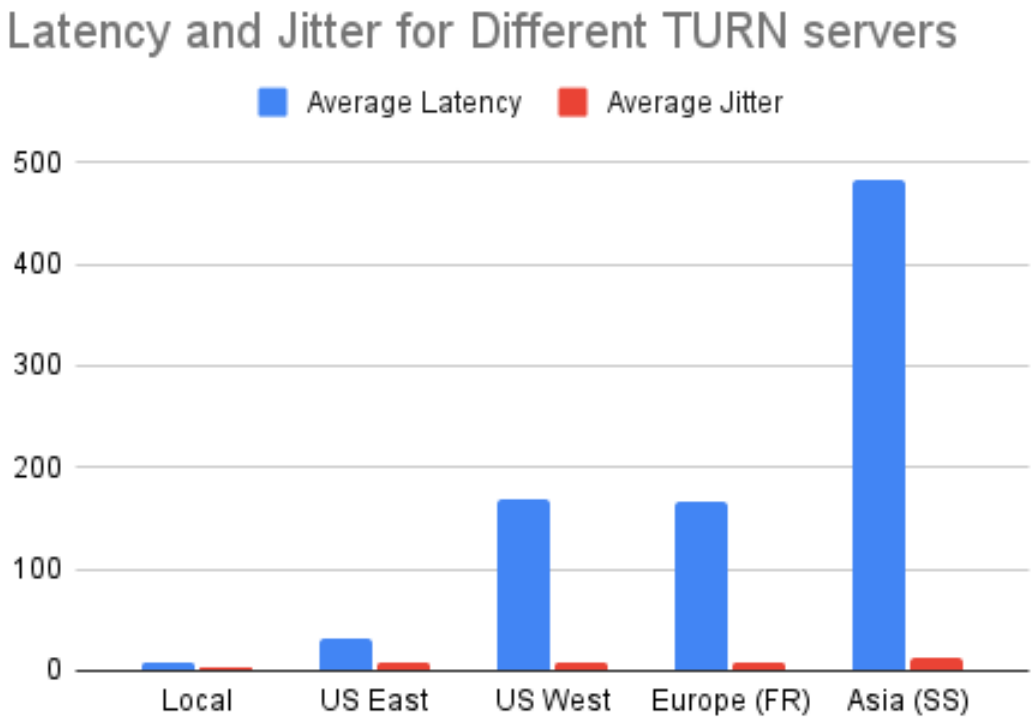

Figure 5-9. Latency and Jitter (measured in ms) for TURN servers located in different regions)

there are still a few observations we can make. We see that baseline latency when an MitM is present includes the effects of δ, while this is not the case if there is no MitM. We can also see how while the baseline during an MitM attack is higher, the actual latency measured is still a function of 2δ. As predicted while the MitM can make the connection seem slower than reality, they cannot remove the artificial latency from appearing in some form.

In Figure 5-7 we look at how our successful our detection of MitMs are. First in the case where there is no MitM, we see that there is on average a high level of accuracy in correctly determining that the channel is free of eavesdroppers. The accuracy does decline slightly over time, likely due to occasional spikes in latency. However, with the observed trend, it is unlikely that accuracy will significantly fall.

In the case where an MitM is present, we see something far more dramatic. There is a rapid increase in accuracy, which grows sharply from 0 before flatting out around 90% correct prediction for the remainder of the call. This seems to indicate that the differences in latency caused by an MitM's presence are significant enough to detect reliably.

Overall, we find that LATENT can detect MitM in practice and provide an additional avenue to authenticate a communication channel.

5.5.4 TURN server analysis

In addition to our experiments run for the LATENT protocol, we wished to also understand what the latency and jitter expectations are like for real world webRTC systems using different TURN servers. This information would better inform the ideal value for the latency parameter δ as well as which settings are viable (i.e. what would it be like for a long range international call?).

In order to do this we made use of the global TURN server infrastructure provided by Xirsys [33]. These are production TURN servers that are actively supporting

webRTC applications. As a result, these servers should provide realistic numbers of what latency and jitter would be like for a deployed application. We then used the same experimental setup as before to measure the baseline round trip latencies experienced, except we varied the TURN server used. For these experiments the test machines are located near US East.

The results of this are detailed in figure 5-9. We see that for local and US East, the amount of round trip latency experienced is quite low and a small δ is probably realistic. However, further locations such as US West and Europe (Frankfurt) result in a larger latency and as a result, require a larger δ in order for it to dominate the total latency measure. We note that since we are assuming both parties to be equidistant to the TURN server, this is roughly double what would be measured if one party was actually located in one of these regions. Finally, we see that the latency is almost 500ms for the Asia (Singapore) TURN server. As it is unlikely that a server this far would be considered equidistant for two parties, we can see this as an upper bound on the existing latency in the network. As calls at this latency would already be difficult to maintain, the LATENT protocol is less viable for calls this long range.

Interestingly, jitter seems to be fairly consistent among all regions, with only a very slight increase from about 7ms to 13ms as we move from Europe to Asia. This is significant as it means that average latency measurements should remain fairly accurate even for small samples. Additionally, it indicates that jitter detection could be a potential secondary avenue to detect MitMs.

As TURN servers are an agreed upon system parameter and can be publicly pinged, having established values of δ based on which TURN server is being used could be a way to have a more adjustable amount of added latency which can fit different communication scenarios while still being robust to adversarial manipulation.

5.6 Related Work

We briefly highlight similar avenues of work to what is presented in this paper.

5.6.1 Other Forced Latency Variants

Forced latency protocols received further attention in [22, 57, 88, 101], however they all have similar limitations. These protocols all attempt to refine Interlock into a more practical protocol, however they still rely on a poorly defined notion of "semantic irregularity" in the conversation contents. Additionally, messages are assumed to be sent sequentially and both parties are not guaranteed the same detection properties. We see our work as specifically addressing these limitations. There is also the TESLA protocol [75], but this protocol does not make any attempt to hide conversation data.

5.6.2 Distance Bounding

Distance bounding is sometimes offered as a solution to detecting MitM [80]. However, these protocols rely on extremely low latency which is not always realistic. Additionally, if the client do not know their relative geographic separation, it is difficult to effectively employ these techniques. In situations where relays are used, a compromised relay would not impact the distance packets travel so a distance bounding approach would not detect its presence.

5.6.3 Captchas and Human-Based Cryptography

The idea of using humans to build cryptography is not new. It was first introduced formally by Naor [69] and has been most popularly applied in the form of Captchas [7, 8, 18, 55, 69, 98] and security ceremonies [78]. These techniques are distinguished by their use of puzzles, which in some instances can be generated by computer, that can only be solved by Humans. While this is a very robust area of research it focuses primarily on offline solvable puzzles and detecting a human in a human to

computer interaction. Our work instead attempts to allow for two humans to mutually authenticate in an online setting that does not require prior setup of puzzles or other functionalities.

There has also been work building systems where a human can verify authentication was done properly through the use of "human perceptible freshness" (HPF) and "human perceptible authenticators" (HPAs) [79]. However, these systems also assume the existence of an out-of-band channel that can perform this authentication, an assumption that our work does not make.

Beyond authentication, there have been other attempts to combine humans and cryptography to produce hybrid crypto-systems [38]. These systems either leverage humans to perform some subtask, or allow humans to carry out some cryptographic functionality without the need of a computer.

5.7 Conclusion

In this work we investigated a framework for constructing secure communications protocols by combining forced-latency with a set of assumptions about humans' ability to detect impersonation. This is, to our knowledge, the first work to provide both a security framework and a set of working protocol tools for this problem. Our work leaves several open problems: at present, we are unable to adaptively select network latencies based on real-world network conditions, as this adaptation opens new opportunities for attackers to manipulate the network. Moreover, it would be valuable to conduct an empirical evaluation of how well our human authentication assumptions hold up under different forms of adversarial manipulation. Finally, it would be interesting to how protocols such as LATENT can be used to bootstrap security in various settings.

Bibliography

[1] Mihir Bellare. "Practice-oriented provable-security." In: *Information Security: First International Workshop, ISW'97 Tatsunokuchi, Ishikawa, Japan September 17–19, 1997 Proceedings 1*. Springer. 1998, pp. 221–231.

[2] Mihir Bellare and Phillip Rogaway. "Entity Authentication and Key Distribution." In: *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '93. Santa Barbara, California, USA: Springer-Verlag, 1994, pp. 232–249. URL: http://dl.acm.org/citation.cfm?id=188105.188164.

[3] Daniel J Bernstein et al. "TweetNaCl: A crypto library in 100 tweets." In: *International Conference on Cryptology and Information Security in Latin America*. Springer. 2014, pp. 64–83.

[4] Bruno Biais et al. *Equilibrium Bitcoin Pricing*. DOI: 10.2139/ssrn.3261063.

[5] Block Chain Analysis. *Block Chain Analysis*. http://www.block-chain-analysis.com/. 2014.

[6] Josh Blum et al. "E2e encryption for zoom meetings." In: *Zoom Video Commun., Inc., San Jose, CA, Tech. Rep. Version* 2.1 (2020).

[7] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. "Identity-based encryption with efficient revocation." In: *Proceedings of the 15th ACM conference on Computer and communications security*. ACM. 2008, pp. 417–426.

[8] A. Boldyreva et al. "Human Computing for Handling Strong Corruptions in Authenticated Key Exchange." In: *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. Aug. 2017, pp. 159–175. DOI: 10.1109/CSF.2017.31.

[9] Dan Boneh and Xavier Boyen. "Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles." In: *EUROCRYPT '04*. Vol. 3027 of LNCS. 2004, pp. 223–238.

[10] Dan Boneh et al. "How relevant is the Turing test in the age of sophisbots?" In: *IEEE Security & Privacy* 17.6 (2019), pp. 64–71.

[11] Nikita Borisov, Ian Goldberg, and Eric Brewer. "Off-the-record Communication, or, Why Not to Use PGP." In: WPES '04. Washington DC, USA: ACM Press, 2004, pp. 77–84. DOI: 10.1145/1029179.1029200.

[12] Yazan Boshmaf, Husam Al Jawaheri, and Mashael Al Sabah. "BlockTag: Design and Applications of a Tagging System for Blockchain Analysis." In: *Proceedings of the 34th IFIP TC11 Information Security Conference & Privacy Conference*. Ed. by Gurpreet Dhillon et al. Lisbon, Portugal: Springer International Publishing, June 2019, pp. 299–313. DOI: 10.1007/978-3-030-22312-0_21.

[13] JP Buntinx. "What is RuffCT and How Will It Affect Monero?" In: *The Merkle* (2017).

[14] Benedikt Bünz et al. "Bulletproofs: Short Proofs for Confidential Transactions and More." In: 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[15] *ByteCoin*. At https://bytecoin.org/.

[16] Jon Callas, Alan Johnston, and Philip Zimmermann. *ZRTP: Media Path Key Agreement for Unicast Secure RTP*. RFC 6189. Apr. 2011. DOI: 10.17487/RFC6189. URL: https://rfc-editor.org/rfc/rfc6189.txt.

[17] Jan Camenisch and Anna Lysyanskaya. "Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials." In: *CRYPTO '02*. Extended Abstract. 2002. URL: http://cs.brown.edu/~anna/papers/camlys02.pdf.

[18] Ran Canetti, Shai Halevi, and Michael Steiner. "Mitigating Dictionary Attacks on Password-Protected Local Storage." In: *Advances in Cryptology - CRYPTO 2006*. Ed. by Cynthia Dwork. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 160–179.

[19] Chainalysis. *Chainalysis Inc*. https://chainalysis.com/. 2017.

[20] John Chan and Phillip Rogaway. "On Committing Authenticated-Encryption." In: 2022, pp. 275–294. DOI: 10.1007/978-3-031-17146-8_14.

[21] Alishah Chator and Matthew Green. "How to squeeze a crowd: reducing bandwidth in mixing cryptocurrencies." In: *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2018, pp. 40–49.

[22] David Chaum. *Distributed Communication Security Systems*. U.S. Patent Application Publication 2006/0218636 A1. Sept. 2006.

[23] David L Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms." In: *Communications of the ACM* 24.2 (1981), pp. 84–90.

[24] W. Diffie and M. Hellman. "New Directions in Cryptography." In: *IEEE Trans. Inf. Theor.* 22.6 (Sept. 1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638. URL: http://dx.doi.org/10.1109/TIT.1976.1055638.

[25] dnaleor. *Warning: DASH privacy is worse than Bitcoin*. Steemit. July 13, 2016. URL: https://steemit.com/bitcoin/@dnaleor/warning-dash-privacy-is-worse-than-bitcoin (visited on 02/07/2020).

[26] Yevgeniy Dodis et al. "Anonymous Identification in Ad Hoc Groups." In: *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*. 2004, pp. 609–626. DOI: 10.1007/978-3-540-24676-3_36. URL: https://doi.org/10.1007/978-3-540-24676-3_36.

[27] Yevgeniy Dodis et al. "Fast Message Franking: From Invisible Salamanders to Encryptment." In: 2018, pp. 155–186. DOI: 10.1007/978-3-319-96884-1_6.

[28] Elliptic. *Elliptic Enterprises Limited*. https://www.elliptic.co/. 2017.

[29] Michael Fröwis et al. "Safeguarding the Evidential Value of Forensic Cryptocurrency Investigations." In: (July 28, 2019). arXiv: 1906.12221.

[30] Yipeng Gao et al. “Research on the Security of Visual Reasoning CAPTCHA.” In: *USENIX Security Symposium*. 2021, pp. 3291–3308.

[31] Christina Garman et al. “Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage.” In: 2016, pp. 655–672.

[32] Mike Gleason. *Microsoft adds end-to-end encryption to Teams*. 2021. URL: https://www.techtarget.com/searchunifiedcommunications/news/252511130/Microsoft-adds-end-to-end-encryption-to-Teams.

[33] *Global TURN Server Cloud Provider - Xirsys - WebRTC*. [Online; accessed 15. Feb. 2023]. Aug. 2022. URL: https://xirsys.com.

[34] Steven Goldfeder et al. “When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies.” In: *Proceedings on Privacy Enhancing Technologies*. Ed. by Rachel Greenstadt, Damon McCoy, and Carmela Troncoso. Vol. 2018. Barcelona, Spain: Sciendo, Oct. 2018, pp. 179–199. DOI: 10.1515/popets-2018-0038.

[35] Opera Google Mozilla. *WebRTC project*. 2007. URL: https://webrtc.org/.

[36] Matthew D. Green and Ian Miers. “Bolt: Anonymous Payment Channels for Decentralized Currencies.” In: *CCS ’16*. Vol. 2016. 2016. URL: http://eprint.iacr.org/2016/701.

[37] Jens Groth and Markulf Kohlweiss. “One-Out-of-Many Proofs: Or How to Leak a Secret and Spend a Coin.” In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*. 2015, pp. 253–280. DOI: 10.1007/978-3-662-46803-6_9. URL: https://doi.org/10.1007/978-3-662-46803-6_9.

[38] Kimmo Halunen and Outi-Marja Latvala. “Review of the use of human senses and capabilities in cryptography.” In: *Computer Science Review* 39 (Feb. 2021), p. 100340. DOI: 10.1016/j.cosrev.2020.100340.

[39] Bernhard Haslhofer, Roman Karl, and Erwin Filtz. “O Bitcoin Where Art Thou? Insight into Large-Scale Transaction Graphs.” In: *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS’16)*. Ed. by Michael Martin, Martí Cuquet, and Folmer Erwin. Leipzig, Germany: CEUR-WS.org, Sept. 13, 2016. URL: http://ceur-ws.org/Vol-1695/paper20.pdf (visited on 06/14/2020).

[40] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. “Blindly Signed Contracts: Anonymous On-Blockchain and Off-Blockchain Bitcoin Transactions.” In: *BITCOIN ’16*. 2016.

[41] Dennis Hofheinz and Eike Kiltz. “Programmable Hash Functions and Their Applications.” In: *CRYPTO 2008*. Ed. by David Wagner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 21–38.

[42] Susan Hohenberger and Brent Waters. “Realizing Hash-and-Sign Signatures under Standard Assumptions.” In: *Advances in Cryptology - EUROCRYPT 2009*. Ed. by Antoine Joux. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 333–350.

[43] Danny Yuxing Huang et al. “Tracking Ransomware End-to-end.” In: *Proceedings of the 39th IEEE Symposium on Security & Privacy (S&P)*. Ed. by Bryan Parno and Christopher Kruegel. San Francisco, CA, USA: Institute of Electrical and Electronics Engineers (IEEE), May 2018, pp. 618–631. DOI: 10.1109/SP.2018.00047.

[44] Tibor Jager et al. “On the security of TLS-DHE in the standard model.” In: *Advances in Cryptology–CRYPTO 2012*. Springer, 2012, pp. 273–293.

[45] Wenzel Jakob. *Lock-free parallel disjoint set data structure*. June 14, 2020. URL: https://github.com/wjakob/dset.

[46] Wenzel Jakob. *pybind11 — Seamless operability between C++11 and Python*. Version 2.5.0. Mar. 31, 2020. URL: https://github.com/pybind/pybind11.

[47] Aaron Johnson et al. “Users get routed: traffic correlation on Tor by realistic adversaries.” In: 2013, pp. 337–348. DOI: 10.1145/2508859.2516651.

[48] Marc Jourdan et al. “Characterizing Entities in the Bitcoin Blockchain.” In: *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. Singapore, Singapore: Institute of Electrical and Electronics Engineers (IEEE), Oct. 2018, pp. 55–62. DOI: 10.1109/ICDMW.2018.00016.

[49] Ari Juels and John G Brainard. “Client puzzles: A Cryptographic countermeasure against connection depletion attacks.” In: *NDSS*. Vol. 99. 1999, pp. 151–165.

[50] Harry A. Kalodner et al. *BlockSci: Design and applications of a blockchain analysis platform*. Arxiv.org. 2017. URL: http://arxiv.org/abs/1709.02489.

[51] Harry A. Kalodner et al. “BlockSci: Design and applications of a blockchain analysis platform.” In: 2020, pp. 2721–2738.

[52] By Leo Kelion. “Deepfake detection tool unveiled by Microsoft.” In: *BBC News* (Sept. 2020). URL: https://www.bbc.com/news/technology-53984114.

[53] Hugo Krawczyk. “HMQV: A High-Performance Secure Diffie-Hellman Protocol.” In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 546–566.

[54] Hugo Krawczyk. “SIGMA: The ’SIGn-and-MAc’ Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols.” In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 400–425. DOI: 10.1007/978-3-540-45146-4_24. URL: https://iacr.org/archive/crypto2003/27290399/27290399.pdf.

[55] Abishek Kumarasubramanian et al. “Cryptography Using Captcha Puzzles.” In: *Public-Key Cryptography – PKC 2013*. Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 89–106.

[56] Adam Langley et al. “The quic transport protocol: Design and internet-scale deployment.” In: *Proceedings of the conference of the ACM special interest group on data communication*. 2017, pp. 183–196.

[57] EF Lanus and EV Zieglar. “Analysis of a Forced-Latency Defense Against Man-in-the-Middle Attacks.” In: *Journal of Information Warfare* 16.2 (2017), pp. 66–IV.

[58] Giulio Malavolta and Dominique Schröder. *Efficient Ring Signatures in the Standard Model*. In ASIACRYPT '17. 2017.

[59] Bill Marczak and John Scott-Railton. "Move fast and roll your own crypto." In: *Report, The Citizen Lab* (2020).

[60] Momina Masood et al. "Deepfakes generation and detection: state-of-the-art, open challenges, countermeasures, and way forward." In: *Appl. Intell.* 53.4 (Feb. 2023), pp. 3974–4026. DOI: 10.1007/s10489-022-03766-z.

[61] Gregory Maxwell. *CoinJoin: Bitcoin privacy for the real world*. Available at https://bitcointalk.org/index.php?topic=279249.0. Aug. 2013.

[62] Frank McSherry, Michael Isard, and Derek Gordon Murray. "Scalability! But at what COST?" In: *Proceedings of the 15th Workshop on Hot Topics in Operating Systems (HotOS XV)*. Ed. by George Candea. Kartause Ittingen, Switzerland: USENIX Association, May 2015. URL: https://www.usenix.org/conference/hotos15/workshop-program/presentation/mcsherry (visited on 06/12/2020).

[63] Sarah Meiklejohn et al. "A Fistful of Bitcoins: Characterizing Payments Among Men with No Names." In: *Proceedings of the 2013 Internet Measurement Conference (IMC)*. Ed. by Krishna Gummadi and Craig Partidge. Barcelona, Spain: Association for Computing Machinery (ACM), Oct. 2013, pp. 127–140. DOI: 10.1145/2504730.2504747.

[64] Ian Miers et al. "Zerocoin: Anonymous Distributed E-Cash from Bitcoin." In: *Proceedings of the 2013 IEEE Symposium on Security and Privacy*. SP '13. 2013.

[65] Yisroel Mirsky and Wenke Lee. "The creation and detection of deepfakes: A survey." In: *ACM Computing Surveys (CSUR)* 54.1 (2021), pp. 1–41.

[66] Sandra Mitrović, Davide Andreoletti, and Omran Ayoub. "ChatGPT or Human? Detect and Explain. Explaining Decisions of Machine Learning Model for Detecting Short ChatGPT-generated Text." In: *arXiv preprint arXiv:2301.13852* (2023).

[67] Malte Möser et al. "An Empirical Analysis of Traceability in the Monero Blockchain." In: *Proceedings on Privacy Enhancing Technologies*. Ed. by Rachel Greenstadt, Damon McCoy, and Carmela Troncoso. Vol. 2018. Barcelona, Spain: Sciendo, June 2018, pp. 143–163. DOI: 10.1515/popets-2018-0025.

[68] S. Nakamoto. "Bitcoin: A peer-to-peer electronic cash system, 2008." In: (2008). URL: http://www.bitcoin.org/bitcoin.pdf.

[69] Moni Naor. "Verification of a human in the loop or Identification via the Turing Test." Oct. 1998.

[70] Shen Noether. "Ring SIgnature Confidential Transactions for Monero." In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 1098. URL: http://eprint.iacr.org/2015/1098.

[71] OpenAI. "ChatGPT: Optimizing Language Models for Dialogue." In: *OpenAI* (Feb. 2023). URL: https://openai.com/blog/chatgpt.

[72] Brian Parno et al. "Pinocchio: Nearly Practical Verifiable Computation." In: *Proceedings of the 34th IEEE Symposium on Security and Privacy*. Oakland '13. 2013, pp. 238–252.

[73] Cristina Pérez-Solà et al. *Analysis of the SegWit adoption in Bitcoin*. URL: https://deic-web.uab.cat/~guille/publications/papers/2018.recsi.segwit.pdf (visited on 06/13/2020).

[74] Cristina Pérez-Solà et al. "Another coin bites the dust: an analysis of dust in UTXO-based cryptocurrencies." In: *Royal Society Open Science* 6.1 (1 Jan. 2019), p. 180817. DOI: 10.1098/rsos.180817.

[75] Adrian Perrig et al. "The TESLA broadcast authentication protocol." In: *Rsa Cryptobytes* 5.2 (2002), pp. 2–13.

[76] Andreas Pfitzmann and Marit Köhntopp. "Anonymity, unobservability, and pseudonymity—a proposal for terminology." In: *Designing privacy enhancing technologies*. Springer. 2001, pp. 1–9.

[77] *Quality of Service Design Overview > QoS Requirements of VoIP | Cisco Press*. [Online; accessed 16. Feb. 2023]. Dec. 2004. URL: https://www.ciscopress.com/articles/article.asp?p=357102.

[78] Kenneth J Radke. "Security ceremonies: including humans in cryptographic protocols." PhD thesis. Queensland University of Technology, 2013.

[79] Kenneth Radke and Colin Boyd. "Security proofs for protocols involving humans." In: *The Computer Journal* 60.4 (2017), pp. 527–540.

[80] Jason Reid et al. "Detecting relay attacks with timing-based protocols." In: *Proceedings of the 2nd ACM symposium on Information, computer and communications security*. ACM. 2007, pp. 204–213.

[81] Ronald L Rivest and Adi Shamir. "How to expose an eavesdropper." In: *Communications of the ACM* 27.4 (1984), pp. 393–394.

[82] Ronald L. Rivest, Adi Shamir, and Yael Tauman. "How to Leak a Secret." In: *ASIACRYPT '01*. Ed. by Colin Boyd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565.

[83] Phillip Rogaway. "Authenticated-Encryption with Associated-Data." In: (2002).

[84] Dorit Ron and Adi Shamir. "Quantitative Analysis of the Full Bitcoin Transaction Graph." In: *Financial Cryptography '13*. 2013.

[85] Nicolas van Saberhagen. *Cryptonote v2.0*. Available at https://cryptonote.org/whitepaper.pdf. Oct. 2013.

[86] Eli Ben Sasson et al. "Zerocash: Decentralized anonymous payments from Bitcoin." In: *IEEE Security and Privacy*. 2014.

[87] Svenja Schroder et al. "When SIGNAL hits the Fan: On the Usability and Security of State-of-the-Art Secure Mobile Messaging." In: *1st European Workshop on Usable Security*. Proceedings of 1st European Workshop on Usable Security. July 2016. DOI: http://dx.doi.org/10.14722/eurousec.2016.23012. URL: http://eprints.cs.univie.ac.at/4799/.

[88] Alan T Sherman et al. "Chaum's protocol for detecting man-in-the-middle: Explanation, demonstration, and timing studies for a text-messaging scenario." In: *Cryptologia* 41.1 (2017), pp. 29–54.

[89] Maliheh Shirvanian and Nitesh Saxena. "Wiretapping via Mimicry: Short Voice Imitation Man-in-the-Middle Attacks on Crypto Phones." In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA: ACM, 2014, pp. 868–879. DOI: 10.1145/2660267.2660274. URL: http://doi.acm.org/10.1145/2660267.2660274.

[90] *Shop with Dash*. 2017. URL: https://www.dash.org/merchants/.

[91] Signal. "What is a safety number and why do I see that it changed?" In: (2023). URL: https://support.signal.org/hc/en-us/articles/360007060632-What-is-a-safety-number-and-why-do-I-see-that-it-changed-.

[92] Tom Simonite. "A Zelensky Deepfake Was Quickly Defeated. The Next One Might Not Be." In: *WIRED* (Mar. 2022). URL: https://www.wired.com/story/zelensky-deepfake-facebook-twitter-playbook.

[93] Iain Stewart et al. "Committing to quantum resistance: a slow defence for Bitcoin against a fast quantum computing attack." In: *Royal Society Open Science* 5.6 (6 June 2018), p. 180410. DOI: 10.1098/rsos.180410.

[94] Shi-Feng Sun et al. "RingCT 2.0: A Compact Accumulator-Based (Linkable Ring Signature) Protocol for Blockchain Cryptocurrency Monero." In: *ESORICS 2017*. Ed. by Simon N. Foley, Dieter Gollmann, and Einar Snekkenes. Cham: Springer International Publishing, 2017, pp. 456–474.

[95] TweetNaCl. *TweetNaCl*. Available at https://tweetnacl.cr.yp.to. 2014.

[96] Serge Vaudenay. "Secure communications over insecure channels based on short authenticated strings." In: *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference*. Lecture Notes in Computer Science 3621 (2005), pp. 309–326. URL: http://infoscience.epfl.ch/record/99433.

[97] Elham Vaziripour et al. "Is that you, Alice? A Usability Study of the Authentication Ceremony of Secure Messaging Applications." In: *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. Santa Clara, CA: USENIX Association, 2017, pp. 29–47. URL: https://www.usenix.org/conference/soups2017/technical-sessions/presentation/vaziripour.

[98] Luis von Ahn et al. "CAPTCHA: Using Hard AI Problems for Security." In: 2003, pp. 294–311. DOI: 10.1007/3-540-39200-9_18.

[99] Brent Waters. "Efficient Identity-Based Encryption Without Random Oracles." In: *EUROCRYPT '05*. Vol. 3494 of LNCS. 2005, pp. 114–127.

[100] *What is StringCT?* [Online; accessed 15. Mar. 2023]. URL: https://monero.stackexchange.com/questions/5997/what-is-stringct/5999#5999.

[101] Zooko Wilcox-O'Hearn. *Defense Against Middleperson Attacks*. Mar. 2003. URL: http://web.archive.org/web/20040804155004/http://zooko.com:80/defense_against_middleperson_attacks.html.

[102] Deressa Wodajo and Solomon Atnafu. "Deepfake video detection using convolutional vision transformer." In: *arXiv preprint arXiv:2102.11126* (2021).

[103] *XMRChain*. Available at https://xmrchain.net/. 2017.

[104] Chen-Zhao Yang et al. “Preventing deepfake attacks on speaker authentication by dynamic lip movement analysis.” In: *IEEE Transactions on Information Forensics and Security* 16 (2020), pp. 1841–1854.

[105] Haaroon Yousaf, George Kappos, and Sarah Meiklejohn. “Tracing Transactions Across Cryptocurrency Ledgers.” In: *Proceedings of the 28th USENIX Security Symposium (USENIX Security)*. Ed. by Nadia Heninger and Patrick Traynor. Santa Clara, CA, USA: USENIX Association, Aug. 2019, pp. 837–850. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/yousaf (visited on 06/13/2020).

[106] Yury Zhauniarovich et al. *Characterizing Bitcoin donations to open source software on GitHub*. July 9, 2019. arXiv: 1907.04002.

[107] *Zoom Acquires Keybase and Announces Goal of Developing the Most Broadly Used Enterprise End-to-End Encryption Offering*. May 2020. URL: https://blog.zoom.us/zoom-acquires-keybase-and-announces-goal-of-developing%5C%5C-the-most-broadly-used-enterprise-end-to-end-encryption-offering/.

[108] *Zoom is malware: why experts worry about the video conferencing platform*. Apr. 2020. URL: https://www.theguardian.com/technology/2020/apr/02/zoom-technology-security-coronavirus-video-conferencing.

[109] *Zoom to pay $85 million to users after lying about end-to-end encryption*. Aug. 2021. URL: https://9to5mac.com/2021/08/03/zoom-to-pay-85-million-to-users-after-lying%5C%5C-about-end-to-end-encryption/.