

Dartmouth College

Dartmouth Digital Commons

Dartmouth College Ph.D Dissertations

Theses and Dissertations

Spring 5-15-2022

Information Provenance for Mobile Health Data

Taylor A. Hardin

Dartmouth College, Taylor.A.Hardin.GR@Dartmouth.edu

Follow this and additional works at: <https://digitalcommons.dartmouth.edu/dissertations>



Part of the [Databases and Information Systems Commons](#), [Information Security Commons](#), and the [Systems Architecture Commons](#)

Recommended Citation

Hardin, Taylor A., "Information Provenance for Mobile Health Data" (2022). *Dartmouth College Ph.D Dissertations*. 79.

<https://digitalcommons.dartmouth.edu/dissertations/79>

This Thesis (Ph.D.) is brought to you for free and open access by the Theses and Dissertations at Dartmouth Digital Commons. It has been accepted for inclusion in Dartmouth College Ph.D Dissertations by an authorized administrator of Dartmouth Digital Commons. For more information, please contact dartmouthdigitalcommons@groups.dartmouth.edu.

INFORMATION PROVENANCE FOR MOBILE HEALTH DATA

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

in

Computer Science

by

Taylor A. Hardin

Guarini School of Graduate and Advanced Studies

Dartmouth College

Hanover, New Hampshire

April, 2022

Examining Committee:

(Chair) David F. Kotz, Ph.D.

Sean W. Smith, Ph.D.

Shagufta Mehnaz, Ph.D.

Fan Zhang, Ph.D.

F. Jon Kull, Ph.D.

Dean of the Guarini School of Graduate and Advanced Studies

Abstract

Mobile health (mHealth) apps and devices are increasingly popular for health research, clinical treatment and personal wellness, as they offer the ability to continuously monitor aspects of individuals' health as they go about their everyday activities. Many believe that combining the data produced by these mHealth apps and devices may give healthcare-related service providers and researchers a more holistic view of an individual's health, increase the quality of service, and reduce operating costs. For such mHealth data to be considered useful though, data consumers need to be assured that the authenticity and the integrity of the data has remained intact—especially for data that may have been created through a series of aggregations and transformations on many input data sets. In other words, *information provenance* should be one of the main focuses for any system that wishes to facilitate the sharing of sensitive mHealth data. Creating such a trusted and secure data sharing ecosystem for mHealth apps and devices is difficult, however, as they are implemented with different technologies and managed by different organizations. Furthermore, many mHealth devices use ultra-low-power micro-controllers, which lack the kinds of sophisticated Memory Management Units (MMUs) required to sufficiently isolate sensitive application code and data.

In this thesis, we present an end-to-end solution for providing information provenance for mHealth data, which begins by securing mHealth data at its source: the mHealth device.

To this end, we devise a memory-isolation method that combines compiler-inserted code and Memory Protection Unit (MPU) hardware to protect application code and data on ultra-low-power micro-controllers. Then we address the security of mHealth data outside of the source (e.g., data that has been uploaded to smartphone or remote-server) with our health-data system, *Amanuensis*, which uses Blockchain and Trusted Execution Environment (TEE) technologies to provide confidential, yet verifiable, data storage and computation for mHealth data. Finally, we look at identity privacy and data freshness issues introduced by the use of blockchain and TEEs. Namely, we present a privacy-preserving solution for blockchain transactions, and a freshness solution for data access-control lists retrieved from the blockchain.

Acknowledgments

This thesis is a testament to the sheer amount of love and support that I have received from numerous colleagues, friends, and family members during my time here at Dartmouth. Undoubtedly, I will fail to acknowledge all who have helped me these last six years, but take that as further proof of the overwhelming number of positive people that I am fortunate enough to have had in my life.

I would like to begin by thanking my advisor, David Kotz, for his guidance, wisdom, and support. Regardless of his various responsibilities and obligations – whether those be serving as the Provost, taking a year-long sabbatical in Switzerland, or serving as the Provost *again* – Dave always found time for me and made sure that I had the resources I needed to succeed. Truly, Dave exemplifies what it is to be a great advisor, scientist, and leader; and for that I am eternally grateful. I would also like to thank my committee members, Sean Smith, Shagufta Mehnaz, and Fan Zhang for their time and expertise during my proposal and defense. Additionally, I would like to thank Jacob Sorber for kick-starting my research career all those years ago as an undergraduate at Clemson University.

I am so very thankful for my friends and for the countless euchre games we've played, the mountains we've skied, and margaritas we've consumed together. In no particular order: Becca Miller, Varun Mishra, Andrew Peterson, James Busch, Alex Zagaria, David Clemens-Sewall, Dan Affsprung, Josie Nordrum, Rachel Osmundsen, Namya Malik, Travis

Peters, Tim Pierson, Pete Brady, Josiah Hester, Nick Wagner, Christian Singleton, John Clark, Matt Cook, and Lisa and Tracy Wallace. Of course, I would be remiss if I did not mention my dog, Xena Beans, who is the sole reason that I have any friends to be thankful for in the first place.

Lastly, I could not have accomplished this without the unwavering love and support of my family. In case I have not expressed this enough, I would like to say to my brother Conner, mother Mona, father James, grandma Rosie, grandma Glenda, grandpa Bill, tio Juan-Ramon, and tia Melissa: I love you.

Contents

Abstract	ii
Acknowledgments	iv
Contents	vi
List of Figures	xi
List of Tables	xii
1 Introduction	1
1.1 Contributions	3
1.2 Prior Work	6
2 Securing Application Code and Data on Ultra-Low-Power Microcontrollers	7
2.1 Contributions	8
2.2 Background	9
2.2.1 Amulet	10
2.2.2 Memory Protection Units (MPUs)	11
2.3 Security Model	11
2.3.1 Adversary Model	12

2.3.2	Security Assumptions	12
2.4	Memory Isolation	13
2.4.1	Memory Map	13
2.4.2	MPU Configuration	15
2.4.3	Memory Accesses	17
2.4.4	Context Switches	18
2.4.5	Amulet Firmware Toolchain (AFT)	18
2.4.6	Memory Isolation Summary	19
2.5	Evaluation	19
2.5.1	Isolation Overhead	19
2.5.2	Benchmark Applications	20
2.6	Related Work	22
2.7	Conclusion	23
3	Designing a Secure and Verifiable Health Data System	24
3.1	Contributions	25
3.2	Background	26
3.2.1	Blockchain	26
3.2.2	Trusted Execution Environments	27
3.3	Security Model	28
3.3.1	Adversary Model	28
3.3.2	Security Assumptions	28
3.4	System Components	30
3.4.1	Blockchain	30
3.4.2	Trusted Execution Environment	32
3.4.3	Smartphones & mHealth Technologies	33
3.4.4	Consumer Applications	34
3.4.5	Off-Chain Data Store	34

3.5	Enclaves	34
3.5.1	Data Storage Enclave	34
3.5.2	Data Processing Enclave	35
3.5.3	Blockchain Interface Enclave	35
3.6	Keys & Key Management	35
3.6.1	Identification Key	36
3.6.2	Signing Key	36
3.6.3	Data Key	36
3.6.4	Sealing Key	37
3.6.5	Master Data Key	37
3.7	Smart Contracts	38
3.7.1	DataSource Smart Contract	38
3.7.2	ConfidentialComputation Smart Contract	40
3.7.3	Registry Smart Contracts	40
3.8	Data Life Cycle	41
3.8.1	Adding mHealth Data	41
3.8.2	Confidential Computations	45
3.8.3	Adding Information	49
3.9	Security Analysis	51
3.9.1	Blockchain Security	51
3.9.2	Enclave Security	53
3.9.3	Off-Chain Data Storage Security	55
3.10	Implementation	56
3.10.1	VeChain Thor Blockchain	57
3.10.2	Intel SGX & GrapheneSGX	57
3.11	Evaluation	58
3.11.1	Blockchain Performance	58

3.11.2	Enclave Performance	61
3.12	Related Work	62
3.12.1	Blockchain-Based Health-Data Systems	62
3.12.2	Confidentiality Preserving Blockchains	63
3.13	Conclusion	64
4	Provenance, Privacy, and Permission in TEE-Enabled Blockchain Data Systems	66
4.1	Contributions	67
4.2	Background	68
4.2.1	Blockchain	68
4.2.2	Trusted Execution Environments	69
4.2.3	Amanuensis	69
4.3	Security Model	71
4.3.1	Adversary Model	71
4.3.2	Security Assumptions	72
4.4	Data Freshness	74
4.4.1	NOVOMODO	75
4.4.2	Freshness Protocol	77
4.5	Identity Privacy	78
4.5.1	Enclave Proxies	79
4.5.2	Smart Contracts	80
4.5.3	Transactions	81
4.6	Information Provenance	83
4.6.1	Provenance Packet Structure	84
4.6.2	Attestation Reports	84
4.7	Security Analysis	86
4.7.1	Data Freshness	86

4.7.2	Identity Privacy	88
4.7.3	Discussion	89
4.8	Evaluation	90
4.8.1	Data Upload Requests	91
4.8.2	Confidential Computation Requests	92
4.8.3	Verifying Information Provenance	92
4.8.4	Executing the Computation Program	94
4.8.5	Overall Performance	95
4.8.6	Observations	97
4.9	Related Work	97
4.10	Conclusion	99
5	Summary and Future Directions	100
5.1	Hardware Improvements in Ultra-Low-Power MCUs	102
5.2	Identity Privacy in Blockchain Networks	102
5.3	Trusted Time Sources	103
5.4	Intel SGX Side-Channels and Deprecation	103
5.5	TEE Optimizations	104
5.6	Confidential Computation Security	105
5.7	Complex Data Access-Control Policies	106
	Bibliography	107

List of Figures

2.1	MPU Memory Diagram	14
2.2	MPU Simulated Test Results	20
2.3	MPU Experimental Test Results	21
3.1	Amanuensis System Overview Diagram	31
3.2	Amanuensis Adding mHealth Data Flow Diagram	43
3.3	Amanuensis Processing mHealth Data Flow Diagram	46
3.4	Amanuensis Adding Information Flow Diagram	49

List of Tables

2.1	MPU Operation Cycle Counts	20
3.1	Amanuensis Key Listing	35
3.2	Amanuensis Symbol List	42
3.3	Amanuensis mHealth Source Capacity	60
3.4	Amanuensis Blockchain Operation Cost	61
3.5	Amanuensis SGX Overhead	62
4.1	Amanuensis Data Upload Time	91
4.2	Amanuensis Data Upload Slowdown	92
4.3	Amanuensis Provenance Verification Time and Slowdown	93
4.4	Amanuensis Computation Program's Execution Time	94
4.5	Amanuensis Computation Request Time	96
4.6	Amanuensis Computation Request Slowdown	96

1

Introduction

Mobile health (mHealth) apps and devices are increasingly popular for health research, clinical treatment, and personal wellness, as they offer the ability to continuously monitor health conditions outside of the traditional healthcare setting. Indeed, mHealth technologies have unlocked a treasure trove of previously unavailable data that many believe will allow healthcare-related service providers and researchers a more holistic view of an individual's health [6, 55, 66, 82, 86, 96]. In turn, providers will be able to more quickly and efficiently identify health issues and prescribe treatments; thereby increasing the quality of service while simultaneously reducing operating costs.

For this belief to become a reality there must first exist a data-management system

with which providers and individuals can securely and privately share their health data. Creating such a system is not without its challenges, however, as mHealth technologies vary and the data they produce may fall under the purview of different parties (e.g., device manufacturers, healthcare providers, private individuals). Given that mHealth data may originate from a multitude of sources that cross organizational and technological boundaries, **data provenance** – the ability to trace the origin of data – is paramount in designing an effective mHealth data-management system. Otherwise, without the ability to verify the authenticity and integrity of data, consumers may not find said data useful. The sensitive nature of health data poses a significant barrier to providing data provenance, however. Health data often contains identifying personal information that must be protected by law (e.g., HIPAA, GDPR), but in many cases it is necessary to reveal this information to prove the validity and authenticity of said data. Furthermore, many mHealth technologies use ultra-low-power micro-controllers (MCUs) to meet the energy and size requirements of always-on mobile devices, but these MCUs lack the kinds of sophisticated Memory Management Units (MMUs) required to sufficiently isolate sensitive application code and data. So, how does one provide provenance for sensitive data produced by insecure devices that are managed by multiple entities while still maintaining data confidentiality and privacy?

Blockchain technology may play a key part in answering this question, because its distributed trust model, transparency, and immutability offer secure and verifiable data trails amidst semi-trusted parties [3, 7, 30, 75, 84, 88]. Health-data systems can use a blockchain to record metadata (e.g., a hash) for health data stored in traditional off-chain databases, record who has access to what data, and record instances of data access. The blockchain ensures that the hash and access records are immutable and visible to all participants so that they can verify the validity and authenticity of health data that is shared with them. While data confidentiality is maintained (since no actual health data is stored on the blockchain), we have only achieved a basic level of data provenance; we can track and verify the movement of health data but not what happens to that data after it has been shared. This

kind of data provenance is not sufficient for complex data ecosystems, where data may be aggregated and transformed to create new data, or *information* [94]. We use the term **information provenance** to refer to the ability to trace the origin of information that may have been derived from a series of aggregations and transformations on many input and intermediary data sets. Additionally, while the distributed nature of the blockchain ensures that transactions *inside* of the network are correctly executed, we still need a mechanism to enforce and verify actions that take place *outside* of the blockchain network if we are to achieve information provenance.

One of the growing sub-fields of blockchain research is that of confidentiality-preserving smart contracts, which leverage techniques like secure multi-party computing, zero-knowledge proofs, and Trusted Execution Environment (TEE) hardware in combination with blockchain technology to create verifiable data trails for sensitive computations performed off-chain. Previously proposed blockchain-based health-data systems are capable of providing *data provenance*, but with confidentiality-preserving smart contract techniques we further their work to achieve *information provenance*.

1.1 Contributions

In this thesis, we present solutions for security and privacy issues related to the collection, storage, and dissemination of sensitive mobile health (mHealth) data. Our initial work (Chapter 2) focuses on securing mHealth data at the source (i.e., on the mobile health device itself). To that end, we devise a memory management technique for securing sensitive mHealth applications and their data. Our memory isolation technique relies on simple Memory Protection Units (MPUs) that are widely available in commercial ultra-low-power microcontrollers (the types of MCUs that are often used in mobile health devices today). We implement and evaluate our solution on the *Amulet* [42], a custom hardware and software platform. We find that with a combination of hardware protection (i.e., MPU enforced

memory access rules) and software memory access checks (injected at compile-time), we are able to completely isolate an application’s code and memory from other programs running on the same mHealth device with a minimal impact on the device’s battery life (less than 0.5% impact).

Next, we shift our focus from the “edge” of the mHealth ecosystem inwards and investigate solutions for securing mHealth data after it has left the mHealth device (Chapter 3). This investigation leads us to the creation of *Amanuensis* [38], a concept for a secure, integrated health-data system that leverages Blockchain and Trusted Execution Environment (TEE) technologies to achieve information provenance for mHealth data.¹ By using a blockchain to record and enforce data-access policies we removed the need to trust a single entity with gate-keeping the health data. Instead, participating organizations form a consortium to share responsibility for verifying data integrity and enforcing access policies for data stored in private data silos. We require that data access and computation take place inside of TEEs, which preserve data confidentiality and provide verifiable attestations that are stored on the blockchain to support information provenance. We also evaluate a prototype implementation of our health-data system on the VeChain Thor blockchain, which we show to be capable of supporting up to 14,256,000 mHealth data sources at \$0.47 per data source per day.

Lastly, we investigate identity privacy and data freshness issues introduced by the use of blockchain and TEEs (Chapter 4). Namely, we present solutions for increasing the privacy of users transacting with the blockchain, and for verifying the authenticity and freshness of access-control lists retrieved from the blockchain. The issue of identity privacy stems from the fact that users on the blockchain are identified via pseudonyms (i.e., hexadecimal character strings), but a naive use of this mechanism leaves room for information leakage. For example, re-using the same pseudonym across transactions (and with different

¹The old term *amanuensis* once referred to someone who is an “intimately trusted servant,” a person responsible for copying what another dictates or for signing documents on behalf of another [27]. In today’s world, this role is often filled by computers – but today’s distributed data systems can not yet be trusted with information provenance. Our system, like a traditional amanuensis, creates distributed copies of records that allow patients and providers to verify how health data is being used and by whom, and to verify the origins of information derived from data.

users/organizations) allows an adversary to identify a user’s on-chain relationships and behaviors. By requiring users to submit transactions through one of the many TEE servers in Amanuensis, however, we can re-write the transactions so that they appear to originate from the TEE server. Thus, user identities are hidden behind these TEEs, or “enclave proxies”, and adversaries with access to the blockchain are no longer able to discern meaningful on-chain relationships.

With regards to data freshness, TEEs can attest to the integrity of data and code inside of the trusted boundary, but they cannot make any guarantees about data received from the outside (e.g., an access control list retrieved from the blockchain database). Furthermore, delivery of messages from outside of the trusted boundary may be arbitrarily delayed by a buggy or malicious operating system (OS). This arbitrary delay (combined with the fact that TEEs do not have access to a trustworthy wall-clock time source) means that TEEs cannot verify the freshness of data received from the blockchain (e.g., that an access control list is the most recent version). By combining NOVOMODO [76], a verification method for digital certificates, with a trusted third-party time service (e.g., NTS-over-TLS), we are able to verify the authenticity and freshness of access-control lists retrieved from the blockchain. Thus, adversaries cannot feed TEEs stale access-control lists or arbitrarily delay delivery of valid lists to gain access to data outside of their allowed time period.

In addition to our work on identity privacy and data freshness, we present an investigation, implementation, and evaluation of our information provenance model under the Intel SGX TEE platform. Specifically, we present a protocol for irrevocably tying confidential programs and their outputs (i.e., information) to Intel SGX attestations for the purpose of information provenance. Given that data produced by mHealth devices and applications are likely to be used with machine-learning models, we train several real-world machine-learning applications to determine the run-time overhead of confidentiality and provenance. We find that each program exhibits a slowdown between 1.1-2.8x when run inside of the Intel SGX environment, and took an average of 59 milliseconds to verify the provenance of input and

output data sets.

1.2 Prior Work

This thesis largely consists of text from our previous publications (shown below).

- Taylor Hardin, Ryan Scott, Patrick Proctor, Josiah Hester, Jacob Sorber, David Kotz. **Application Memory Isolation on Ultra-Low-Power MCUs.** In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, July 2018. (Chapter 2)
- Taylor Hardin, David Kotz. **Blockchain in Health-Data Systems: A Survey.** In *Proceedings of the IEEE International Symposium on Blockchain Computing and Applications (BCCA)*, October 2019. (Chapter 3)
- Taylor Hardin, David Kotz. **Amanuensis: Information Provenance for Health-Data Systems.** In *Information Processing & Management Special Issue: Blockchain for Information Systems Management and Security*, March 2021. (Chapter 3)
- Taylor Hardin, David Kotz. **Provenance, Privacy, and Permission in TEE-Enabled Blockchain Data Systems.** Accepted to *IEEE International Conference on Distributed Systems (ICDCS)*, April 2022. (Chapter 4)

2

Securing Application Code and Data on Ultra-Low-Power Microcontrollers

Information provenance – the ability to trace the origin of information that may have been derived from a series of aggregations and transformations on many input and intermediary data sets – is dependant on a “root of provenance” with which proofs of authenticity and integrity can be built. For mHealth data, this “root of provenance” is the mHealth device; as the creator, the mHealth device must guarantee the authenticity and integrity of the data it produces. Otherwise, any data derived from these original mHealth data sets cannot be proved to be genuine.

In this chapter, we introduce our memory-management scheme that provides data security and confidentiality for mHealth applications running on multi-tenant ultra-low-power mHealth devices (i.e., devices that support the concurrent execution of multiple applications). Note that our memory-management solution provides data integrity and confidentiality, but not authenticity. It remains the responsibility of mHealth application developers to use cryptographic techniques (e.g., digital signatures) to ensure the authenticity of data produced by mHealth devices. By providing developers with confidentiality and integrity, however, our memory-management technique prevents the modification or disclosure of sensitive secrets (e.g., keys) used for data authentication by other buggy or malicious software running on the mHealth device. Thus, by securing the memory of mHealth devices, we have given developers the ability to create a “root of provenance” for information provenance.

2.1 Contributions

Our work focuses on a fundamental security property: *memory isolation*, which ensures that no application can read, write, or execute memory locations outside its own allocated region, or call functions outside a designated system API. To this end, we present a memory isolation technique, which leverages compiler inserted code and a low-sophistication Memory Protection Unit (MPU) found in many microcontrollers, to achieve better performance than software-only memory isolation counterparts. We use the open-source Amulet platform [42] to implement and evaluate the following isolation methods: (1) compiler-enforced language limitations (no pointers, no recursion), (2) compiler-inserted run-time memory isolation (address-space bounds verification), and (3) MPU-supported memory isolation (hardware enforced failure). The first option is the approach taken by the Amulet team, which limits the programmer to a subset of C. Pointers are disallowed, and the compiler inserts code for run-time bounds-checking on arrays. In the second approach, we modify the Amulet implementation to allow for pointers and recursion, but our custom compiler inserts code to

validate each pointer dereference to ensure the application stays within its bounds. In the third approach we implement a new combination, in which the OS and compiler coordinate the dynamic assignment of the MPU’s limited functionality – and limited compiler-inserted pointer checking – to enable the desired isolation. In summary, we make the following contributions:

1. An analysis of design considerations, including security issues, that enable multiple applications on ultra-low-power wearables, with minimal burden on the programmer or the user;
2. A memory-isolation technique, using the limited-function hardware memory protection unit (MPU) found in commodity ultra-low-power microcontrollers, combined with compile-time analysis of application code, to sandbox application code and memory;
3. A prototype implementation as a refinement of the open-source Amulet platform [42];
4. An evaluation that compares the performance of the Amulet platform’s limited language-based memory-isolation mechanism, a full-featured software-only approach, and a full-featured MPU-assisted mechanism.

2.2 Background

Some mHealth devices are capable of running a variety of apps, including third-party apps installed by the user. The battery life for multi-application mHealth devices, however, tends to be far shorter than their single-purpose (application) counterparts. To balance these trade-offs, some mHealth devices (such as the Amulet [42]) seek to achieve the battery life of single purpose devices, while still maintaining the capability to run multiple third-party apps, by using ultra-low-power microcontrollers (MCUs). One of the main challenges with using ultra-low-power MCUs in multi-application contexts is that they lack the hardware-based memory-protection mechanisms – such as Memory Management Units (MMU) – needed to

ensure that applications cannot interfere with each other’s sensitive code and data. Instead of an MMU, many ultra-low-power MCUs provide a Memory Protection Unit (MPU), which only allows the user to configure read/write/execute permissions for a few distinct regions of memory. The lack of adequate memory protection mechanisms makes it difficult to provide long battery life *and* strong security properties that allow multiple third-party apps to coexist on ultra-low-power mHealth devices. We begin with an overview of the Amulet platform on top of which our implementation is built. Then we detail the capabilities, and shortcomings, of MPUs and their effects on memory isolation.

2.2.1 Amulet

We apply our memory-isolation technique to the latest open-source build of Amulet¹, which implements memory isolation through compiler-enforced language limitations (no pointers, no recursion, no goto statements and no inline assembly). The Amulet system allows an Amulet user to select a customized mix of applications to run on her Amulet wristband, from a suite of applications developed independently by separate app developers. The Amulet system consists of three core parts – AmuletOS, Amulet Runtime, and the Amulet Firmware Toolchain (AFT). AmuletOS provides the core system services and an event-based scheduler that drives the apps’ state machines, delivering events by calling the appropriate event-handler function with parameters representing the details of the event. Amulet Runtime provides a state-machine environment in which all applications run. The Amulet Firmware Toolchain (AFT) [42], analyzes, transforms, merges, and compiles the user’s desired applications with the AmuletOS to construct a firmware image for installation on the user’s Amulet device.

¹The latest open-source release of the Amulet platform can be found at <https://github.com/AmuletGroup/amulet-project>

2.2.2 Memory Protection Units (MPUs)

Many ultra-low-power MCUs are equipped with a basic Memory Protection Unit, but they have some or all of the following shortcomings: (1) they do not provide enough memory distinct regions to sandbox each application; (2) they do not protect certain segments of memory (e.g., hardware registers, RAM); and (3) they have arcane protection boundary rules, because they depend on opaque hardware implementations.

Amulet devices use a TI MSP430FR5969 MCU [49], which does not have support for virtualizing memory; meaning that all applications (and the OS) running on the MCU share a single address space. To protect this address space, the MSP430FR5969 provides an MPU that is capable of dividing the memory space into three distinct regions², each with their own set of read, write, and execute permissions. Unfortunately, these MPU-protected memory regions only apply to main memory (addresses 0x4400 and above), and not to SRAM, peripheral registers, or the interrupt vector table. Furthermore, there is no notion of kernel and user space; all code running on the MSP430FR5969 is allowed to modify the MPU settings regardless of whether it is system code or application code.

2.3 Security Model

Maintaining data confidentiality and integrity is paramount for mHealth devices, as the data they produce is highly sensitive. Thus, while we wish to remove some of the restrictions and burdens placed on developers by existing memory-isolation techniques, we need to ensure that we do so while still maintaining strong security guarantees for applications running on these ultra-low-power mHealth devices. Below we outline the adversary model and security assumptions that our memory-isolation technique is designed against.

²Technically, the MSP430FR5969 supports four distinct memory regions, but the fourth region is bounded to the InfoMem segment whose bounds do not move.

2.3.1 Adversary Model

Given that our focus is on memory-isolation for multi-application mHealth devices, we consider an adversarial application that wishes to gain access to the private health data and/or code of another application running on the same mHealth device for the purpose of (1) learning something sensitive about one or more individual(s), or (2) modifying private health data to pass it off as authentic. We assume that an adversarial application may try to gain access to memory outside of their own context through one of the following methods:

1. By using a specific pointer to access a disallowed memory location.
2. By sending a specific pointer to a system function, causing the system function to indirectly access a disallowed memory location.
3. By writing to the stack irresponsibly, causing a stack overflow, and the potential for harm.

2.3.2 Security Assumptions

Our aim for memory isolation is to prevent application-based attacks on code and data. In other words, we consider attacks carried out via hardware our malicious OS code to be out-of-scope. We assume the following about the mHealth device platform and OS:

1. the hardware MPU operates correctly, and enforces the memory region boundaries and permissions that we set
2. our OS (AmuletOS) is unmodified and correctly loaded onto the target device along with the applications
3. our compile-time tool used to rewrite applications is unmodified and operates correctly

2.4 Memory Isolation

Typically, memory isolation is enforced via a memory-management unit (MMU), which is a piece of hardware that translates *virtual* addresses into *physical* addresses. This abstraction (i.e., memory virtualization) is used by operating systems to provide applications with the illusion that they have a contiguous address space all to themselves. In reality, their code and data is often intermingled with other applications. Since ultra-low-power MCUs do not have MMUs the applications they run are able to directly address (and thus read or modify) other application’s code and data. The lack of an MMU is why the designers of Amulet chose to enforce memory-isolation via language limitations (e.g., disallowing pointers). Our memory-isolation technique relaxes these language restrictions by changing the way applications (and the OS) are laid out in memory, such that a limited-capability memory protection unit (MPU) in combination with run time software checks are sufficient for isolating applications on ultra-low-power MCUs. Below we detail our new memory layout for the Amulet device, as well as how the MPU is configured to enforce memory isolation.

2.4.1 Memory Map

At compile time, we use the Amulet Firmware Toolchain (AFT) to calculate the size of each application’s code and data. With this knowledge, then AFT then lays out each application in memory so that there are contiguous memory segments containing application code, followed by their data, and finally followed by the application stack (Figure 2.1 details this new layout). Recall that the MPU in the Amulet device only protects the persistent memory (i.e., FRAM), but does not apply to the other types of memory in the MCU. Namely, the MPU will not prevent instructions from reading or writing the peripheral registers, InfoMem, SRAM, or interrupt vectors. For this reason, we provide each application with their own stack in FRAM so that we may use the MPU to ensure no other application is able to read or

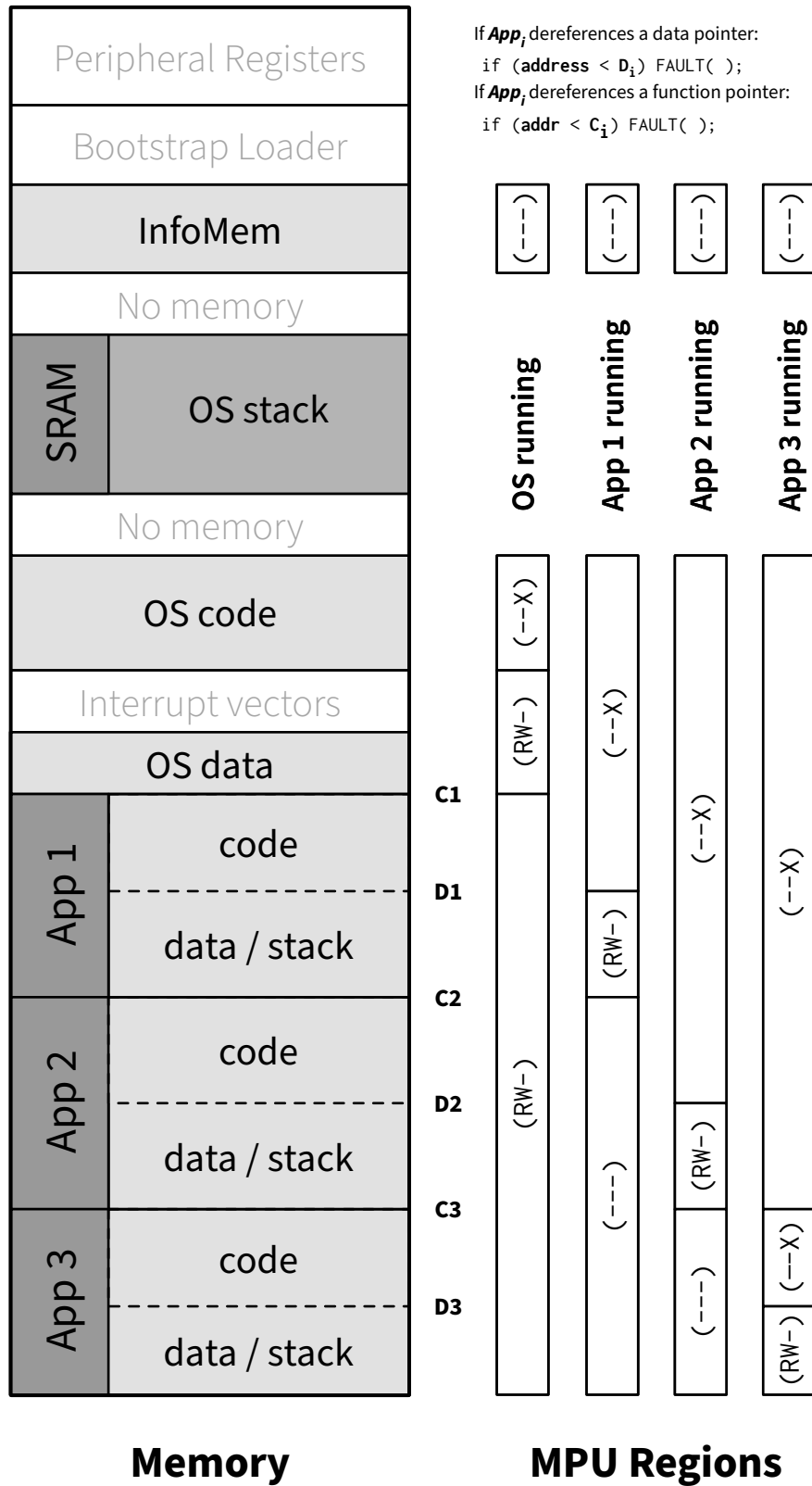


Figure 2.1: Memory diagram of our approach, and MPU regions per application.

modify their stack (and vice-versa).

Additionally, the MPU only allows us to subdivide the FRAM memory into three distinct segments (each with their own r/w/x permissions). To properly isolate applications, however, we need at least four MPU-controlled segments: app code, app data/stack, off-limits memory below the app, and off-limits memory above the app. As such, the MSP430 MPU only has the ability to prevent accesses above the application's higher memory boundary but not below its lower boundary. Thus, to prevent malicious applications from accessing memory below their lower boundary, we use the AFT to identify unsafe memory operations at compile-time and wrap them with run-time memory checks. In addition to adding run-time checks for preventing memory accesses below the application's boundary, the AFT also inserts code for re-configuring the MPU settings and stack pointer during a context switch (e.g., switching to AmuletOS code during a system call).

2.4.2 MPU Configuration

Use of the MPU requires a different memory mapping than in the original Amulet implementation. We leverage the SRAM for the AmuletOS stack, the low FRAM for AmuletOS code and data, and the high FRAM for app code and data, grouped by app.³ Each app's code and data are separated, with its code in lower addresses than its data. The MPU has four segments, of which we can make good use of three.⁴ InfoMem, the first segment, is fixed to a certain address range and its configuration can be changed any time by any code. Furthermore, only two boundaries are adjustable: the boundary between segment 1 and 2 and the boundary between segment 2 and 3.

To allow application developers to use C pointers, we leverage previously described MPU hardware. Consider, as an example, Application 2 in Figure 2.1. While Application 2

³If the AmuletOS is too large to fit in the low FRAM, it could span the interrupt vectors, but for simplicity we do not show it as such in the diagram.

⁴MPU segment 0 is pinned to the InfoMem, which is only 512 bytes and which we currently do not use. We anticipate using the InfoMem in future revisions, for a return-address stack that protects the return address from stack overflow bugs and attacks.

is running, we configure the MPU segments as follows:

1. InfoMem (unused; no access);
2. OS, lower-memory apps, and current-running app's code (execute-only);
3. current-running app's data and stack (read-write only);
4. higher-memory apps (no access).

All of the app's code is gathered in one region, all of its data and stack in another region. The MPU configuration triggers a fault if a stray pointer references anything in higher regions (shown as Application 3 in the figure), but the MPU cannot fully protect regions in addresses below the application's code segment. When Application 2 makes a call to AmuletOS, we first reconfigure the MPU segments as follows:

1. InfoMem (unused; no access);
2. OS code (execute-only);
3. interrupt vectors and OS data (read-write only);
4. apps (read-write only).

This configuration allows the AmuletOS to run its own code and, as needed, to manipulate data in both the app and OS regions.

It's important to note an important design change from Amulet as it was originally introduced. The Amulet system uses a single stack – shared by both the OS and the current application. This approach is possible because Amulet uses a run-to-completion approach, so there is no need to retain a stack for non-running apps. It is also possible because app code cannot use pointers, and thus cannot read any memory outside its statically allocated global variables, or outside its current stack frame. If we were to stick with the same single-stack model, we would need to `bzero` the stack region every time we switched apps, lest the new

app glean information from the stack tailings left behind by the prior app. We chose instead to allocate a distinct region of memory for each app's stack, removing this cost (and other costs to ensure stack references remain in-bounds) at the cost of increased memory usage.

That brings us to another important design decision related to security and the application stack. Languages such as C traditionally place a function's return address on the stack, and jump indirectly through that address as part of the function-return instruction. Stack overflows in buggy or malicious code can overwrite that entry on the stack, however, causing the function to return to a different address. We leverage the compiler to insert code to bounds-check the return address before every function return. Furthermore, we place the top of the app stack below the app's data in the app's data/stack segment, and allow the stack to grow downward. The compiler and linker can compute the size of the app's data region, and estimate the maximum stack depth, to ensure the data/stack segment is large enough for the app's needs. If the app overflows its stack, for example by too-deep recursive calls, it will cross an MPU boundary into an execute-only code region and trigger a fault.

2.4.3 Memory Accesses

An important role for the run-time system is to handle application faults; when the app attempts an invalid memory access, it jumps to a `FAULT` function to log app-specific information about the fault. At compile time, the AFT uses its transformation tools to verify that the app only calls approved API functions and reads approved system global variables, and to insert code that verifies (at run-time) every pointer dereference before it occurs. Notice that every one of these checks is a simple comparison against a constant, followed by a conditional branch (jump) to the fault-handling code. Because all app code is processed by the AFT, and the app cannot inline any of its own assembly code, the resulting code is guaranteed to check every pointer used by the app.

2.4.4 Context Switches

The AmuletOS provides an API for applications to access utilities and system services. We need to swap MPU configurations and change stacks on each transition, and we need to carefully handle application-provided pointers passed through API calls to the OS. Furthermore, because each app, and the OS, has a separate stack segment, we need to change the stack pointer on every transition between the OS and an app. Thus, the code to adjust the MPU configuration must also tackle the stack-switch. The compiler wraps an app's event handler with assembly code to initialize the app's stack and stack pointer upon jumping to the app code. In addition the compiler inserts assembly code to create a stack frame on the OS stack and set the stack pointer before jumping to the API function inside the OS.

2.4.5 Amulet Firmware Toolchain (AFT)

We extend the Amulet Firmware Tool (AFT) to implement the MPU and software-only method checks previously described. These tasks are accomplished by the AFT in a four-phase code analysis. In the first phase, the AFT checks for any still unsupported language features – such as inline assembly and GOTO statements. In addition, the AFT enumerates each memory access and OS API call on an app by app basis. Examination of the application call graph and the stack frame for each function determines the maximum stack size for each app. In the event of recursion, the maximum stack size cannot be determined and the AFT cannot guarantee a large enough stack to prevent overflow. During the second phase, the MPU configuration code and the previously mentioned memory access checks (with placeholder values for app boundaries) are injected into the code. The third phase marks apps with memory section attributes for the linker, as well as injecting the assembly code needed to manipulate the stack pointer. The last phase involves determining the code size of each app, updating the linker script to place each app in high memory (as detailed in Figure 2.1), and updating the memory access checks from phase two with the correct app boundaries. The AFT completes by recompiling the modified code into the final firmware

image.

2.4.6 Memory Isolation Summary

To summarize, the MPU memory isolation method consists of configuring the MPU for an app and inserting lower-bound checks, while the compiler inserted (software-only) method mentioned earlier consists of not using the MPU and inserting both an upper and lower memory bound check. Although the MSP430’s MPU itself is not sufficient to protect the system and other applications from pointer misuse by a buggy (or malicious) app, it is useful: in our approach, we strategically leverage both the MPU and the compiler to accomplish the necessary protections. This section details the memory map used for MPU, as well as how we handle memory accesses and context switches.

2.5 Evaluation

In this section we evaluate the costs of application isolation. Our proposed system allows developers to write pure C, instead of a constrained Amulet C, enabling them to more easily write (or port) application code to the Amulet platform. We look at the isolation overhead of a large set of Amulet applications for three methods in Section 2.5.1, and see that while the overhead of our isolation method is higher than a feature-limited Amulet C, the impact of the overhead on battery lifetime is negligible. In Section 2.5.2 we describe three benchmark applications, and the trade-offs they display between computation-intensive and OS-intensive applications.

2.5.1 Isolation Overhead

We use the Amulet Resource Profiler (ARP) and the ARP-view tool to count the number of memory accesses and context switches per state and transition, per application. Using ARP-view, we can account for the rate of environmental, user, and timer events set by the

Operation	No Isolation	Feature Limited	MPU	Software Only
Memory Access	23	41	29	32
Context Switch	90	90	142	98

Table 2.1: Average cycle count for basic memory isolation operations.

developer, combine this information with the counted number of memory accesses and context switches, and extrapolate the number of cycles of overhead for isolating applications. We can then convert the estimated cycles into energy cost (in Joules) to estimate the negative impact of isolation on battery lifetime. The results of this experiment are shown in Figure 2.2 for nine applications that are part of the Amulet platform. These applications comprise thousands of lines of code, and many have been deployed in user studies [11, 12]. **For all applications, isolation using either the MPU or Software Only methods has less than a 0.5% impact on battery lifetime.**

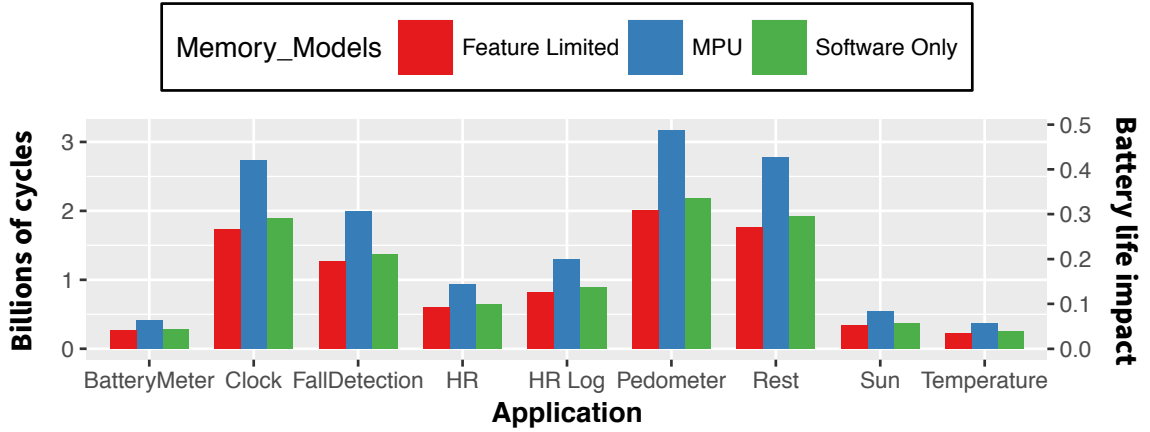


Figure 2.2: Isolation overhead in billions of cycles per week, and battery lifetime impact percentage for a variety of applications. Gathered using the Amulet Resource Profiler infrastructure.

2.5.2 Benchmark Applications

We further explore the system overhead of application isolation through several benchmark applications with varying levels of memory accesses. We designed a **Synthetic App** a

simple application whose purpose is to test the two fundamental actions that incur memory-protection overheads: *memory accesses* and *context switches*. We then investigate two major functions in our **Activity Detection App**, which correspond to *Activity Case 1* and *Activity Case 2* in Figure 2.3. These functions have a high number of memory accesses compared to context switches. Finally, we design a **Quicksort App**: an application that runs the quicksort algorithm with a high number of memory accesses and no context switches. Each application was run 200 times and a hardware timer on the MSP430FR5969 MCU was used to measure the time of each iteration (with a precision of 16 cycles).

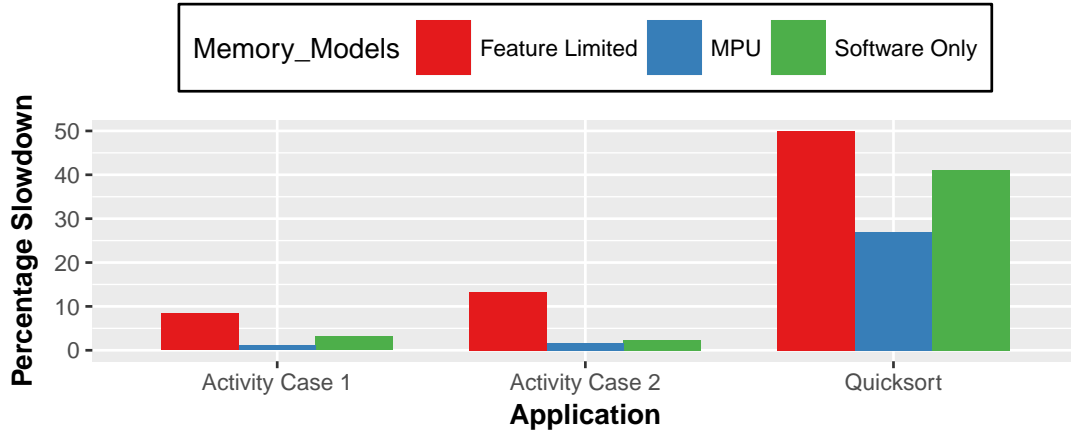


Figure 2.3: Percentage slowdown for each memory isolation method calculated by comparing them to running apps with no isolation method.

The results from the synthetic app test in Table 2.1 show that our MPU method had the fastest memory accesses, but the slowest context switches. This result was expected, and validates the simulation results, as our method only requires half the number of bounds checks as the Software Only approach, but incurs extra overhead for reconfiguring the MPU during context switches. Figure 2.3 further confirms the results from Table 2.1, which is that our method is the most effective when used for computationally heavy applications.

2.6 Related Work

Early operating systems for mobile devices like TinyOS [59] and others [10, 28, 36] reduced complexity [57], enabled dynamic reprogramming [58], and provided interfaces for concurrent execution [29]. These platforms did not provide memory isolation, however, nor did they allow installation of multiple third-party applications. Recent work on memory isolation for ultra-low-power MCUs rely on techniques such as language restrictions, compile-time analysis, and binary-code rewriting.

AmuletOS [42] uses a dialect of ANSI C, termed AmuletC, which disallows pointers and recursion. TockOS [60] writes kernel code in Rust, a type-safe and memory-safe language, and isolates their apps using an MPU. While language modifications can make compile-time analysis easier [91], they tend to limit expressiveness and are rarely enough to ensure complete application isolation. Language features are often coupled with compiler checks, binary-code rewriting, or system-implemented dynamic checks. For example, AmuletOS has a compiler that inserts run-time bounds-checking code around all array accesses [42]. Deputy [20, 21] enforces type safety at compile time; Harbor [56], built on top of SOS [36], rewrites binary code to check any pointer reference and function call. T-Kernel [35] modifies code at load time to secure application memory. Each of these compile- and run-time techniques come with limitations: static (compile-time) techniques depend on language features (or modifications) and clear OS rules, while dynamic (run-time) techniques incur run-time overhead to check memory accesses. Other systems virtualize the single memory space to isolate applications, like Mat  [57], or rely on novel hardware mechanisms such as a Secure Loader hardware unit between the CPU, peripherals, and RAM [53].

MPU-equipped microcontrollers have existed for decades, but in that time MPU capabilities have largely remained the same. Indeed, there are a number of works that propose new MPU architectures [39, 43, 61, 71, 72, 83, 95] to meet the needs of resource-constrained multi-application embedded devices because many current MPU implementations do not. These new MPU designs include support for a larger number of memory segments (e.g., 8,

16), privilege levels (i.e., kernel vs. user), or protection of peripheral configuration registers. That said, there are systems that were able to achieve memory-isolation with existing MPU architectures [2, 51, 87, 108, 111], but they rely on more robust (and power-hungry) 32-bit microcontrollers which do not meet the ultra-low power constraints of Amulet.

Given these prior techniques, our memory-isolation solution takes a new approach that leverages the meager capabilities of resource-constrained MPUs with the lessons learned from years of software-based methods.

2.7 Conclusion

In this chapter, we explore the challenge of memory isolation on ultra-low-power microcontrollers, which offer primitive hardware support for memory protection. Traditional approaches use a range of language limitations, compiler analysis, or dynamic checks (inserted by compiler or other tools); few have leveraged the capabilities of emerging MPUs. Our solution employs MPU hardware to protect most regions of memory from inappropriate access by application code. Our proof-of-concept implementation is limited by the capabilities of the MSP430 MPU, which cannot protect the region below the current app's allocation; thus, the compiler still needs to insert some code for bounds checks – albeit half as many as in the software-only solution. While our approach leveraging the MPU was not effective for apps that make frequent API calls, our MPU isolation approach had, for all applications, less than 0.5% impact on battery lifetime.

3

Designing a Secure and Verifiable Health Data System

In Chapter 2 we began our investigation into information provenance by securing mHealth data at its point of origin: the mHealth device. In this chapter, we shift our focus to ensuring the provenance of data that has left the device. Specifically, we are interested in data that may be shared with one or more intermediate parties for the purpose of transforming and combining it with other data sets to create *new* data (i.e., information). Naively, one could solve the provenance problem by simply creating a public record of all data, who accessed it, and for what purpose. Given the sensitive nature of mHealth data, however, such a solution

would not work in the “real world”. Indeed, there are often legal frameworks in place (e.g., HIPAA, GDPR) to prevent (and punish) the unlawful disclosure of health data. Thus, any data-management solution that provides information provenance also needs to provide data confidentiality if it wishes to be adopted in real-world applications.

3.1 Contributions

We present *Amanuensis*, a concept for a secure, integrated health-data system that leverages Blockchain and Trusted Execution Environment technologies to achieve information provenance for mHealth data. By using a blockchain to record and enforce data-access policies we remove the need to trust a single entity with gate-keeping the health data. Instead, participating organizations form a consortium to share responsibility for verifying data integrity and enforcing access policies for data stored in private data silos. We require that data access and computation take place inside of TEEs, which preserves data confidentiality and provides a verifiable attestation that can be stored on the blockchain in support of information provenance. We evaluate a prototype implementation of Amanuensis – built using Intel SGX trusted execution hardware and the VeChain Thor blockchain platform – which shows that Amanuensis is capable of supporting up to 14,256,000 mHealth data sources at \$0.47 per data source per day.

We begin by presenting background information on Blockchain and Trusted Execution Environments (Section 3.2), as well as related works on blockchain-based health-data systems and confidentiality techniques for blockchains (Section 3.12). Then we define our adversarial and security assumptions (Section 3.3) before introducing the technological components that comprise Amanuensis (Section 3.4). We provide further details on technical make-up and inner-workings our TEE (Section 3.5, Section 3.6) and blockchain (Section 3.7) components, as well as how these components work together to provide information provenance (Section 3.8). Finally, we present a security analysis of our system design (Section 3.9),

and an evaluation (Section 3.11) of our prototype implementation (Section 3.10).

3.2 Background

We begin with an overview of Blockchain and Trusted Execution Environment technology, and define relevant terms that are pertinent to understanding our system design.

3.2.1 Blockchain

While there exist several types of blockchain paradigms, we only present background information for blockchains based on the Ethereum model as that is the type of blockchain used in Amanuensis. A **blockchain** is an append-only database of time-stamped records that are cryptographically linked to protect them from tampering and revision [40]. Unlike traditional databases that are managed by a central authority, a blockchain is a distributed database managed by a number of semi-trusted participants. Users submit **transactions** to the blockchain network, which either contain new data to be recorded or invoke smart contracts (defined below). The servers that maintain the blockchain are referred to as **miners**, and are responsible for validating these transactions and periodically bundling groups of transactions into a block that is cryptographically linked to the prior block. Over time, these linked blocks form a chain – hence the name *blockchain*. Each miner maintains and updates its own local copy of the blockchain via a **consensus algorithm**, which ensures that the majority of the nodes agree on exactly what transactions are being added to the blockchain and in what order. Once a block is added to the blockchain, the details of its transactions are visible to all participants of the network and cannot be modified or removed. Along with logging basic transactions (e.g., Bob pays Alice 10 Bitcoin) some blockchains support **smart contracts**, which allow developers to encode arbitrary logic that is uploaded to, and executed by, the miners in the blockchain network. In effect, a smart contract is a permanent, immutable collection of functions that all nodes execute upon the arrival of certain types of

transactions. Due to the transparent nature of blockchains, all data that passes through the blockchain (whether it is input, intermediate, or output) is visible to those participating in the network and therefore not confidential.

3.2.2 Trusted Execution Environments

The notion of Trusted Execution Environments (TEEs) has been around for decades, yet there is still no wide agreement on a definition for TEEs or their security properties. Further adding to the confusion, the terms *Enclave* or *Secure Enclave* are sometimes used to refer to a TEE while other times may simply refer to a Trusted Application running on a TEE. For the purposes of this thesis, we define Trusted Execution Environments and Enclaves as follows. An **Enclave** is an application that is meant to be executed in a Trusted Execution Environment. The Enclave contains sensitive code and/or data that the developer wishes to protect from the surrounding system. A **Trusted Execution Environment** is a secure computing environment that is resistant to privileged software attacks (i.e., those from an OS or hypervisor) and lightweight physical attacks (i.e., those on hardware outside of the CPU). The Trusted Execution Environment guarantees the authenticity and confidentiality of Enclave code and its run-time states, as well as the confidentiality of Enclave code and data stored off-chip in persistent memory [90]. Furthermore, Trusted Execution Environments provide Enclaves with several trusted-computing functionalities such as data sealing, integrity measurement, remote attestation, and secrets provisioning. **Data sealing** is the process of encrypting Enclave data for persistent storage outside of the TEE. The data-sealing key is unique to the TEE platform and Enclave, thus ensuring that encrypted data is “bound” to that specific Enclave. **Integrity measurement** is performed by the TEE when first loading Enclave code to ensure that the code has not been tampered with while stored in persistent memory. **Remote attestation** is a process for proving to parties outside of the TEE platform (i.e., code executing on a different machine) that an Enclave has been unmodified and is running on a genuine TEE platform. **Secrets provisioning** is a method

for establishing a secure communication channel between an enclave and an outside party; this method is bound to the remote attestation process to ensure that the remote party only shares secrets with a specific enclave that has been attested.

3.3 Security Model

Amanuensis aims to preserve the confidentiality, integrity, and provenance of sensitive health data. Data confidentiality, integrity, and provenance must be upheld for both source data *and* derived information (i.e., information created through a series of aggregations and transformations of source data). Below we present the capabilities afforded to an adversary and the security assumptions for each of the technological components of Amanuensis.

3.3.1 Adversary Model

We consider an adversary that (1) wishes to gain access to private health information to learn something sensitive about one or more individual(s), (2) wishes to modify private health information, or (3) wishes to pass off false health information as authentic. We assume the adversary can (1) gain physical access to a TEE node and remotely compromise the OS, (2) gain control over some small¹ portion of the blockchain network, and (3) arbitrarily read data from the blockchain and off-chain data storage

3.3.2 Security Assumptions

While some of the following security assumptions may seem implicit based on the technological definitions presented in Section 3.2, we choose to explicitly state them to clearly define the scope of security challenges that our system aims to address. Furthermore, we use these assumptions as the basis for our security analysis in Section 3.11. The security of

¹Blockchain networks that use Proof-of-Work (PoW) consensus algorithms can operate correctly if the majority of the mining nodes act in good faith.

any system depends on both its design *and* implementation, but the focus of this chapter is evaluating the design of our system. Hence, security challenges arising from implementation choices (e.g., side-channel attacks on Intel SGX) are largely left out-of-scope.

Blockchain Amanuensis uses a blockchain to maintain the integrity of the data it stores and of the smart-contract operations it performs. For these properties to hold, we must assume that the consensus algorithm is capable of preventing attackers from modifying the blockchain even if they control some miners in the blockchain network. In the context of our implementation, we assume that the majority of the VeChain Authority Nodes act in good faith when executing the consensus protocol.

Trusted Execution Environments Amanuensis has several secure applications, or *Enclaves*, that are responsible for storing, retrieving, modifying, and sharing mHealth data. These Enclaves rely on TEEs to provide a secure and confidential computing environment for handling sensitive mHealth data. We assume that the TEE hardware and firmware are designed such that they guarantee the authenticity of the executed code, the integrity of the run-time states, and the confidentiality of the Enclave’s code and data while preventing outside entities (OS, hypervisor, applications) from gaining access or control. Furthermore, we assume that the TEE hardware has the capabilities for correctly and securely providing the trusted computing features described in Section 3.2 (e.g., data sealing, integrity measurement, remote attestation, and secrets provisioning). We also consider side-channel² and Denial-of-Service (DoS) attacks out-of-scope.

Smartphone We assume that an individual’s smartphone correctly and securely aggregates data from the individual’s mHealth apps and devices and sends the data batches to one of the Enclaves for storage. In this context, the smartphone correctly executes our health-data-

²In concept, TEEs protect against various forms of physical attack, but in reality some implementations have been shown to be vulnerable to side-channel attacks, which leverage observable physical traits of a computing system to infer what operations it is performing and/or what data it is handling.

collection application and has secure storage for protecting encryption and signing keys and sensitive health data.

mHealth Apps and Devices We assume that mHealth apps and devices, regardless of type or manufacturer, operate correctly; they produce accurate data and they can securely communicate with a smartphone. We also assume that they have ways to authenticate the user of the device so as not to produce data for the wrong data subject. Physical attacks on mHealth apps and devices are out-of-scope.

Off-Chain Data Storage We assume nothing about off-chain storage services other than they can be trusted to correctly write data blobs to storage and return the correct data blob for a given storage index.

Cryptography We assume that the underlying cryptographic primitives on which our system is built (e.g., cryptographic hashes, digital signatures, encryption algorithms) are not breakable in any practical time frame.

3.4 System Components

Amanuensis is comprised of several major technological components (pictured in Figure 3.1) that work together to provide confidentiality-preserving information provenance. Below we introduce each component and describe its role in Amanuensis.

3.4.1 Blockchain

The backbone of Amanuensis is its consortium-based blockchain, which provides a suite of smart contracts that allow individuals to share and manage their mHealth data in a confidential yet verifiable fashion. The consortium is comprised of health-related organizations (e.g., hospitals, clinics, insurance providers, public-health agencies, researchers, government

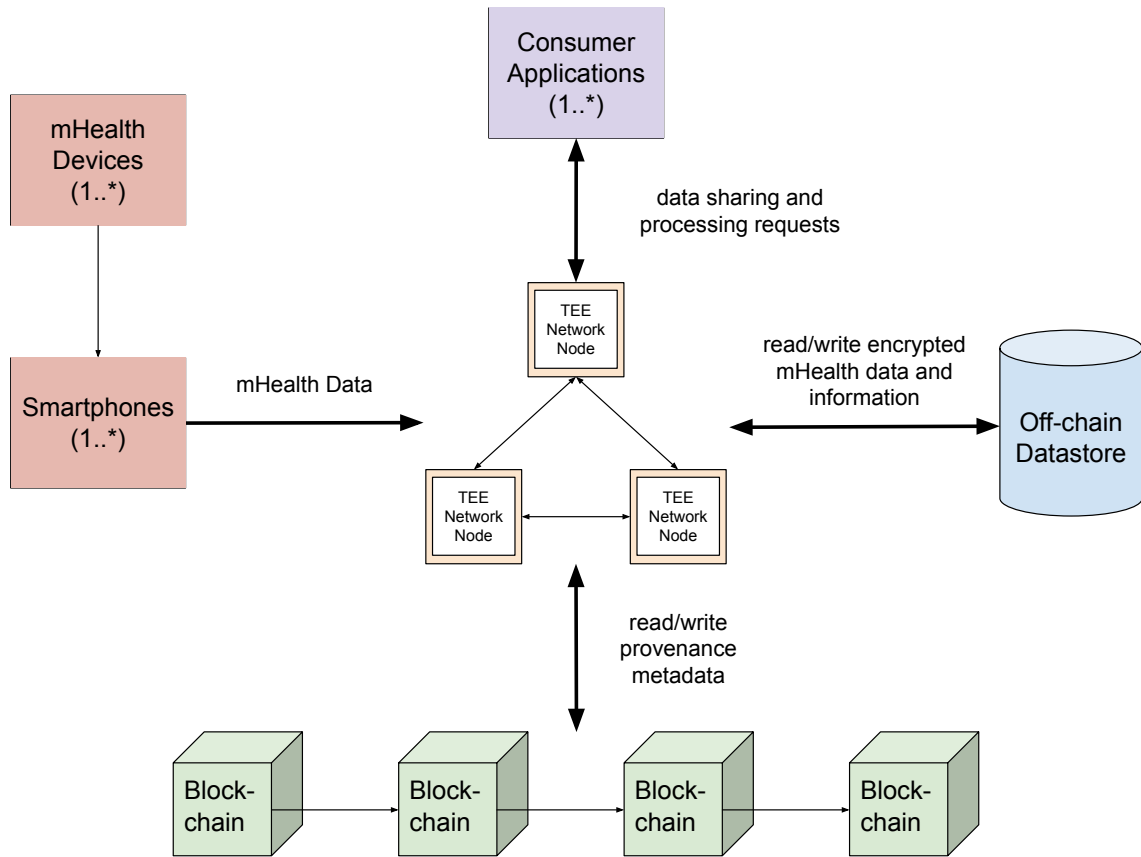


Figure 3.1: **System Overview.** A high-level depiction of the relationship between the different components of the Amanuensis system, where ‘1..*’ denotes one or more.

regulators) that collectively manage the blockchain. This approach eases inter-organization information sharing, because they can use the blockchain to verify the authenticity and validity of shared data. As a consortium blockchain, permission to execute, validate, and add transactions to the blockchain is reserved for miner nodes operated by the consortium members, but, as the contents of the blockchain are public, individuals can still read data from the blockchain and submit transactions they wish to be executed. Currently, individuals trust (often due to a lack of other options) health-service providers to manage their private health data, but they have no way to verify that their data is being handled correctly. By forcing healthcare providers to use a publicly-readable blockchain for data-access decisions, individuals (or watchdogs like consumer-protection organizations) are afforded a look into how sensitive data is managed. The exact number of nodes required to run the blockchain component of Amanuensis is implementation-specific, and thus left up to the members of the health consortium; the number of nodes depends on many factors, including how many participants are in the network, availability, cost, etc.. All our design for Amanuensis requires is that each consortium member participates in the blockchain network in some capacity so as to ensure the availability of the network and to maintain the distributed and immutable nature of the blockchain.

3.4.2 Trusted Execution Environment

Amanuensis also maintains a distributed network of TEE servers to ensure the availability of trusted applications, or *Enclaves*, which are used by individuals and organizations to manage and share their health data. Similar to the blockchain mining nodes, TEE nodes are maintained by members of the blockchain consortium (not by individuals). Amanuensis requires all actions upon mHealth data to occur inside a TEE to ensure that data remains confidential and that actions can be verified. To this end, Amanuensis provides several Enclaves that facilitate secure and confidential creation, storage, retrieval, modification, and dissemination of mHealth data and information.

By using the TEE’s “remote attestation” feature, enclaves can prove (to relevant individuals and organizations) that they are legitimate Amanuensis Enclaves executing on TEE hardware. Attestation reports produced by enclaves, along with other relevant metadata, are stored on the blockchain to create a transparent record of data creation, storage, sharing, and modification that can be verified by data owners and consumers without the need to provide them access to the source data. While provenance metadata is stored in the blockchain, the actual health data is encrypted and stored in off-chain storage services. Storage services may be owned and managed by different service providers, but the keys used to encrypt health data reside in protected TEE memory and on the blockchain (in encrypted form). Thus, once stored in the system, mHealth data can only be decrypted by one of our secure enclaves running in our TEE network. Again, like the blockchain component, the number of TEE nodes is implementation-specific and left up to the members of the health consortium. Ideally, they would choose a number that is sufficient to guarantee availability of the system and to ensure that the TEE component remains distributed amongst the consortium members. In other words, there should not be a single, centralized, TEE node or cluster that is responsible for granting access to the blockchain network and off-chain datastore.

3.4.3 Smartphones & mHealth Technologies

Mobile-health technologies can be used to monitor aspects of an individual’s health as they go about their everyday life and activities. These apps and devices produce streams of data (e.g., raw sensor data, user input, inferences) that are uploaded to an individual’s smartphone. The smartphone serves two purposes: (1) to act as a gateway for the storage of mHealth data in the Amanuensis health-data system, and (2) to create a root of trust that can later be used to authenticate and verify data produced by an individual’s mHealth app or device. We present the details of this “root of trust” in Section 3.8.

3.4.4 Consumer Applications

Amanuensis serves as a back-end data management solution for applications that wish to consume mHealth data for the purpose of providing healthcare-related services. The design and implementation of such applications is orthogonal to the design of Amanuensis; consumer applications should adhere to the protocol laid out in Section 3.8 if they wish to be compatible with Amanuensis, but outside of that we treat them as black boxes.

3.4.5 Off-Chain Data Store

Similar to consumer applications, we treat the off-chain data store as a black-box component. The actual design and implementation of the off-chain data store does not effect our system design, as long as it is capable of storing a data blob and returning an ‘index’ that can later be used to retrieve that same blob. It is an immutable data store, and needs no other database features (like indexing, searching, or manipulating data).

3.5 Enclaves

Amanuensis’ distributed network of TEEs runs instances of three different Enclaves: *Data Storage*, *Data Processing*, and *Blockchain Interface*. Users interact with these enclaves to store, retrieve, manipulate, and share their mHealth data and derived information. Brief descriptions for each Enclave are provided below, with detailed breakdowns of the Enclaves’ functionalities and their interactions in Section 3.8.

3.5.1 Data Storage Enclave

The *Data Storage Enclave (DSE)* is used to store and retrieve mHealth data in the Amanuensis system. More importantly, the DSE is responsible for collecting and validating provenance metadata every time mHealth data is stored or read from storage.

3.5.2 Data Processing Enclave

The *Data Processing Enclave (DPE)* is a secure application for executing confidential smart contracts in the TEE. This application communicates with the Data Storage application (DSE) when it needs to retrieve input for a computation or to record the computation's output.

3.5.3 Blockchain Interface Enclave

The *Blockchain Interface Enclave (BIE)* is a secure application that is used by the Data Processing and Data Storage enclaves to interact with the blockchain. The BIE handles reading and writing data to and from the blockchain, invoking smart contracts, and verifying permissions for data requests.

3.6 Keys & Key Management

Amanuensis relies on asymmetric keys for identifying and authenticating users and data sources, as well as symmetric keys for protecting the confidentiality of sensitive health data. Below we describe these keys in detail. For quick reference, a list of keys and their symbols can be found in Table 3.1.

Table 3.1: Keys used in Amanuensis.

Name	Symbol	Type
Identification Key	PK	Asymmetric
Signing Key	SK	Asymmetric
Data Key	K	Symmetric
Sealing Key	S	Symmetric
Master Data Key	K_{MASTER}	Symmetric

3.6.1 Identification Key

We use the term *Identification Key* to refer to the public component of an asymmetric key pair, (PK, SK) . Identification keys are denoted PK_{NAME} , where $NAME$ is the name of the entity the key belongs to (e.g., a health app, health device, confidential computation, or an individual). As the name suggests, Identification keys are used to identify a particular individual or entity in the Amanuensis system. Furthermore, each Identification key has verification function, denoted $PK()$, that is used to verify a signature created by the corresponding Signing key in the pair (PK, SK) .

As part of the blockchain protocol, all users (e.g., individuals, consortium members) are given asymmetric key pairs that are used to identify and authenticate themselves when transacting on the blockchain. We assume that all mHealth devices and applications come with baked-in unique asymmetric key pairs that can be used to identify and authenticate data sources.

3.6.2 Signing Key

We use the term *Signing Key* to refer to the private component of an asymmetric key pair, (PK, SK) . Signing keys are denoted SK_{NAME} , where $NAME$ is the name of the entity the key belongs to (e.g., a health app, health device, confidential computation, or an individual). As the name suggests, Signing keys provide a function, denoted $SK()$, that allow the owner of a Signing Key to sign (i.e., authenticate) data.

3.6.3 Data Key

We use the term *Data Key* to refer to a symmetric encryption key that is used to encrypt data streams. Data keys are denoted K_{NAME} , where $NAME$ is the name of device or application that produced the data stream. Data keys provide a function for encrypting/decrypting data, denoted $K()$. Data keys are used by the Data Storage Enclaves to encrypt incoming mHealth

data streams before writing them to off-chain storage. We store Data Keys on the blockchain so that they persist and are accessible to all DSE instances; when necessary, any TEE server may service the mHealth data source. Storing Data keys in plaintext on the blockchain would make them available to anyone, of course, so Data keys are first encrypted with the Master Data Key (described below) before being written to the blockchain.

3.6.4 Sealing Key

We use the term *Sealing Key* to refer to a symmetric encryption key that is unique to, and can only be derived by, a specific enclave running on a specific TEE platform. Sealing keys are denoted S_{NAME} , where $NAME$ is the name of the relevant enclave (e.g., DSE, DPE, or BIE). Sealing keys provide a function for encrypting/decrypting data, denoted $S()$. Sealing keys are used by enclaves to encrypt “secrets” (e.g., cryptographic keys and sensitive data) that should persist between enclave execution instances. For example, if an enclave is provisioned with a secret that it wishes to reuse the next time it is instantiated it can “seal” that secret and write it to persistent storage. The next time the enclave is instantiated it can re-derive its Sealing Key and decrypt the secret. In Amanuensis, Enclaves use their Sealing Key to encrypt the Master Data Key so that they continue to have access to it after being stopped and restarted.

3.6.5 Master Data Key

We use the term *Master Data Key* to refer to the symmetric encryption key that is used to encrypt and decrypt Data Keys stored on the blockchain. Whereas there are multiple Identification, Signing, Data, and Sealing keys, there is only one Master Data Key in the Amanuensis System. The Master Data Key is denoted K_{MASTER} , and its function for encrypting/decrypting data is denoted $K_{MASTER}()$. During a one-time setup phase, the Master Data Key is generated by one DSE, which then registers itself with the *EnclaveRegistry* contract described in Section 3.7. Subsequent DSEs that join the network can then look-up

existing enclaves on the registry and retrieve the Master Data Key through the remote attestation and secrets provisioning mechanisms provided by TEEs.

3.7 Smart Contracts

Amanuensis uses a blockchain to track off-chain health data, provenance metadata, data-access policies, mHealth apps and devices, and confidential computation binaries. These data are organized and accessed via smart contracts, which are programs that are stored on, and executed by, the miners in the blockchain network. Much like traditional programs, smart contracts have variables, functions, and persistent memory. The Amanuensis blockchain provides five distinct smart contracts for interacting with the blockchain: *DataSource*, *ConfidentialComputation*, *SourceRegistry*, *ComputationRegistry*, and *EnclaveRegistry*. We describe each contract below.

Algorithm 1 DataSource smart contract

```

1: struct record  $t = \{hash, index, key, attestationDSE\}$ 
2: var dataSourceID
3: var dataSourceOwner
4: mapping( $timestamp \rightarrow record_t$ ) records
5: mapping( $address \rightarrow bool$ ) permissions
6:
7: function grantDataAccess( $consumer$ )
8: function revokeDataAccess( $consumer$ )
9: function hasDataAccess( $consumer$ )
10: function addData( $timestamp, hash, index, key$ 
11:                 $attestationDSE$ )
12: function getData( $timestamp$ )

```

3.7.1 DataSource Smart Contract

The DataSource contract (Algorithm 1) records information for identifying a particular mHealth data source (e.g., mHealth application or device), the data that it produces, and for setting data-access policies. On the blockchain, there is an instance of the DataSource

contract for every mHealth data source in the Amanuensis system. There are two entities that directly interact with this contract: the data source owner and the Blockchain Interface Enclave (BIE). Data source owners make calls to the *grantDataAccess* and *revokeDataAccess* functions (via a smartphone or desktop application) to grant, or revoke, access to their data – by adding or removing a data consumer’s blockchain identity to or from the *permissions* mapping variable. The BIE serves as a proxy for the other Enclaves (Data Storage and Data Processing) when they wish to interact with the blockchain. Some of these interactions involve adding and retrieving provenance metadata to and from DataSource contracts, as well as verifying data-access permissions for a given data consumer. Every time a new batch of data is added to the system (i.e., saved in off-chain storage), provenance metadata is added to the DataSource contract associated with the device that produced the batch of data. To track provenance metadata, the DataSource contract makes use of a struct *record.t* comprised of (1) a hash of the mHealth data batch, (2) an index that can be used to locate that batch in off-chain storage, (3) an encrypted form of the key that was used to encrypt the batch of mHealth data, and (4) an attestation report generated by the Enclave that stored the data.

Algorithm 2 ConfidentialComputation smart contract

```

1: struct record.t = {hashes, indexes, key, appID
2:                   attestationDSE, attestationDPE}
3: var computationID
4: var computationOwner
5: string computationBinary
6: mapping(timestamp → record.t) records
7: mapping(address → bool) permissions
8:
9: function grantDataAccess(consumer)
10: function revokeDataAccess(consumer)
11: function hasDataAccess(consumer)
12: function addData(timestamp, hash, indexes, key
13:                  appID, attestationDSE, attestationDPE)
14: function getData(timestamp)

```

3.7.2 ConfidentialComputation Smart Contract

Functionally similar to the *DataSource* contract, the *ConfidentialComputation* contract (Algorithm 2) records information for identifying confidential programs, their resulting data (i.e., information), and for setting data-access policies. On the blockchain, there is an instance of the *ConfidentialComputation* contract for every confidential program in the Amanuensis system. There are two entities that directly interact with this contract: the computation owner and the Blockchain Interface Enclave (BIE). The information maintained by the *ConfidentialComputation* contract consists of (1) a public identifier of the confidential program's owner, (2) an encrypted binary of the confidential program, (3) a list of data hashes, (4) a list of storage indexes, (5) a list of attestation reports, (6) a list of authorized data consumers, and (7) a list of input data. The binary is stored on the blockchain to ensure that it never changes and remains tied to the data that it produced and a list of input data storage indexes are kept to provide informational-level provenance (i.e., you can tie the resulting data of a confidential computation to all of its input data, which themselves have attestations and proofs of provenance). While the confidential program's binary is stored on the blockchain, it is never actually executed by nodes in the blockchain network (the way that smart contracts are). Instead, the encrypted binary is sent to an enclave where it is decrypted and executed to ensure that the computation and the data it uses remains confidential. The details of the process are described in Section 3.8.

3.7.3 Registry Smart Contracts

There are three registry contracts – *SourceRegistry*, *ComputationRegistry*, and *EnclaveRegistry* – that make it easy locate specific smart contracts and enclaves. Whereas there is a different smart contract instance for every mHealth data source and each confidential program, there is only one instance of each registry contract. These registry contracts should be deployed by an admin account managed by the blockchain consortium. When users deploy a smart contract instance for their health data source or confidential program, they

register it with the *SourceRegistry* or *ComputationRegistry* (respectively) so that our Enclave applications can find its contract when performing data storage, retrieval, and modification operations. The *EnclaveRegistry* is populated by the blockchain consortium and contains a list of IP addresses of active Amanuensis TEE servers on which users can interact with our Enclaves.

3.8 Data Life Cycle

In our system, all derived health information can ultimately be traced back to its original source, i.e., to data generated by mHealth apps and devices. We need to guarantee that data produced by these mHealth technologies are stored and shared in a secure but verifiable way if we wish to provide information provenance for subsequent modifications and aggregations made to the source data. Below we detail how we provide confidential but verifiable information provenance by describing the three major data-flows in our system: Adding mHealth Data, Confidential Computations, and Adding Information. Table 3.2 provides a quick-reference list of symbols we use.

3.8.1 Adding mHealth Data

Our data life cycle begins with mHealth technologies and the streams of health data that they produce. Amanuensis provides users with the ability to securely store, share, and verify such data streams, but the data must first be added to the system. Imagine a scenario in which a data subject, Bob, owns an mHealth device, *HD*, and a smartphone, *SP*. Bob's device produces health data and uploads it to his smartphone, which serves as a gateway to the Amanuensis system. The process for uploading a batch of data, *D*, generated by *HD* is depicted in Figure 3.2 and described in detail below. All inter-entity communications are carried over pre-established secure channels that provide confidentiality, integrity, and reliability (for data in transit) using standard Internet techniques, e.g., TLS over TCP/IP; for

Table 3.2: List of symbols used in Section 3.8.

Symbol	Meaning
A	attestation report
APP	mHealth application
BIE	Blockchain Interface Enclave
C	batch of mHealth data (ciphertext)
CC	confidential computation
D	batch of mHealth data (plaintext)
DPE	Data Processing Enclave
DSE	Data Storage Enclave
H	hash
HD	mHealth device
I	storage index
J	Data Key (ciphertext)
K	Data Key (plaintext)
K_{MASTER}	Master Data Key
L	list of input data batches
M	mHealth data message
P	provenance packet
PK	Identification Key
R	data processing request result
SIG	signature
SK	Signing Key
SP	smartphone

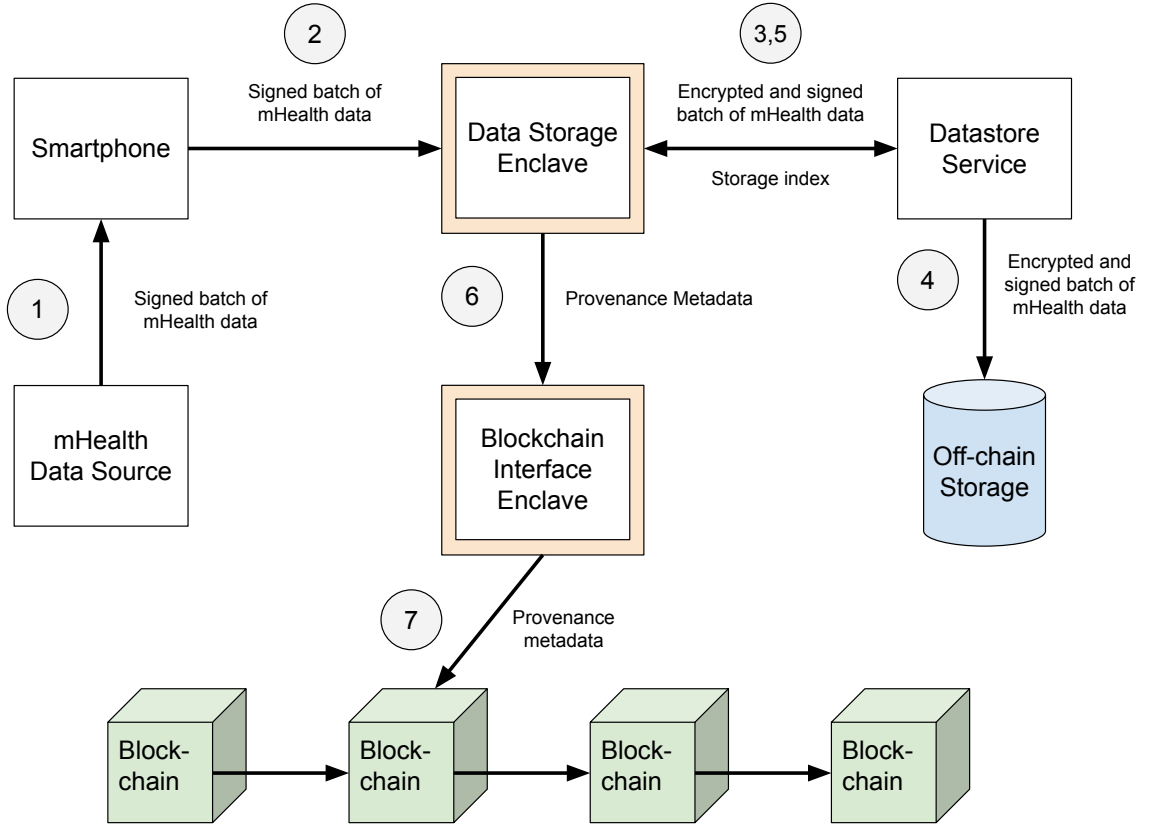


Figure 3.2: **Adding mHealth Data.** After being generated, mHealth data is sent to a smartphone, which serves as a gateway to the Amanuensis system. The smartphone uploads the mHealth data to a Data Storage Enclave, which encrypts and saves the data to off-chain storage. Afterwards, a Data Storage Enclave records provenance metadata on the blockchain so that future data consumers can verify the authenticity and integrity of mHealth data stored in off-chain storage.

brevity, the details of these connections are not shown. Furthermore, while only a single enclave instance (e.g., Data Storage, Data Processing, Blockchain Interface) is depicted in our figures, there exist a number of TEE nodes in Amanuensis that each host one or more enclave instances.

1. HD creates a signature, SIG_{HD} , to identify itself as the creator of the health data, D_{HD} . Then, HD appends its signature to the batch of health data to create a message,

M_{HD} , which it sends to Bob's smartphone.

$$SIG_{HD} = SK_{HD}(D_{HD}) \quad (3.1)$$

$$M_{HD} = D_{HD} || SIG_{HD} \quad (3.2)$$

2. Upon receiving M_{HD} , Bob's smartphone creates a signature using Bob's Signing Key, SK_{BOB} , to indicate that Bob is the subject of the data. Then, SP appends Bob's signature to M_{HD} to create an updated message, M_{SP} , which it sends to a Data Storage Enclave, DSE .

$$SIG_{BOB} = SK_{BOB}(M_{HD}) \quad (3.3)$$

$$M_{SP} = M_{HD} || SIG_{BOB} \quad (3.4)$$

3. DSE encrypts M_{SP} with the Data Key, K_{HD} , which is unique to the data stream produced by HD . The encrypted data, C_{HD} , is then sent to an off-chain Datastore Service for storage.

$$C_{HD} = K_{HD}(M_{SP}) \quad (3.5)$$

4. The Datastore Service treats C_{HD} as a blob and writes it to off-chain storage.
5. The Datastore Service returns a storage index, I_{HD} , to DSE that indicates the location of C_{HD} .
6. After receiving I_{HD} , DSE generates an attestation report,³ A_{DSE} , and calculates a hash, H_{HD} , of M_{SP} . Then, DSE encrypts K_{HD} with the Master Data Key, K_{MASTER} ,

³TEE hardware mechanisms generate a report containing integrity measurements and signatures that are bound to a specific enclave and TEE platform. This report can then be submitted to the TEE vendor for verification, though Amanuensis does not perform this last verification step until the next time this specific batch of health data is used.

to produce J_{HD} . Finally, DSE bundles the provenance information into a packet, P_{HD} , and sends it to a Blockchain Interface Enclave, BIE .

$$H_{HD} = h(M_{SP}) \quad (3.6)$$

$$J_{HD} = K_{MASTER}(K_{HD}) \quad (3.7)$$

$$P_{HD} = A_{DSE} || PK_{HD} || I_{HD} || H_{HD} || J_{HD} \quad (3.8)$$

7. BIE adds P_{HD} to the smart contract associated with HD .

For efficiency, the mHealth data source, or the smartphone, may choose to upload data records through this pipeline in batches. For example, a pulse oximeter might batch its readings once per minute for upload to the smartphone, which may further batch the data once per hour for upload into the Data Storage Service. The protocols above apply equally well to such batches; the selection of batch intervals is application-dependent.

3.8.2 Confidential Computations

Users of Amanuensis can grant others permission to access and/or process their health data and information, but instead of sending these “data consumers” a copy of the data we require them to perform their computations using a Data Processing Enclave. Since a Data Processing Enclave is executed in a TEE, we can ensure the confidentiality of sensitive health data while still providing the consumer proof that their computations were performed correctly. For some data consumer’s application, APP , we depict the process of retrieving and computing upon data in Figure 3.3 and describe the steps in detail below.

1. APP creates a data-processing request consisting of its Identification key, PK_{APP} , the Identification Key of the computation it would like to execute, PK_{CC} , and a list of indexes, L , that are used to locate the input health data and information sets. Then

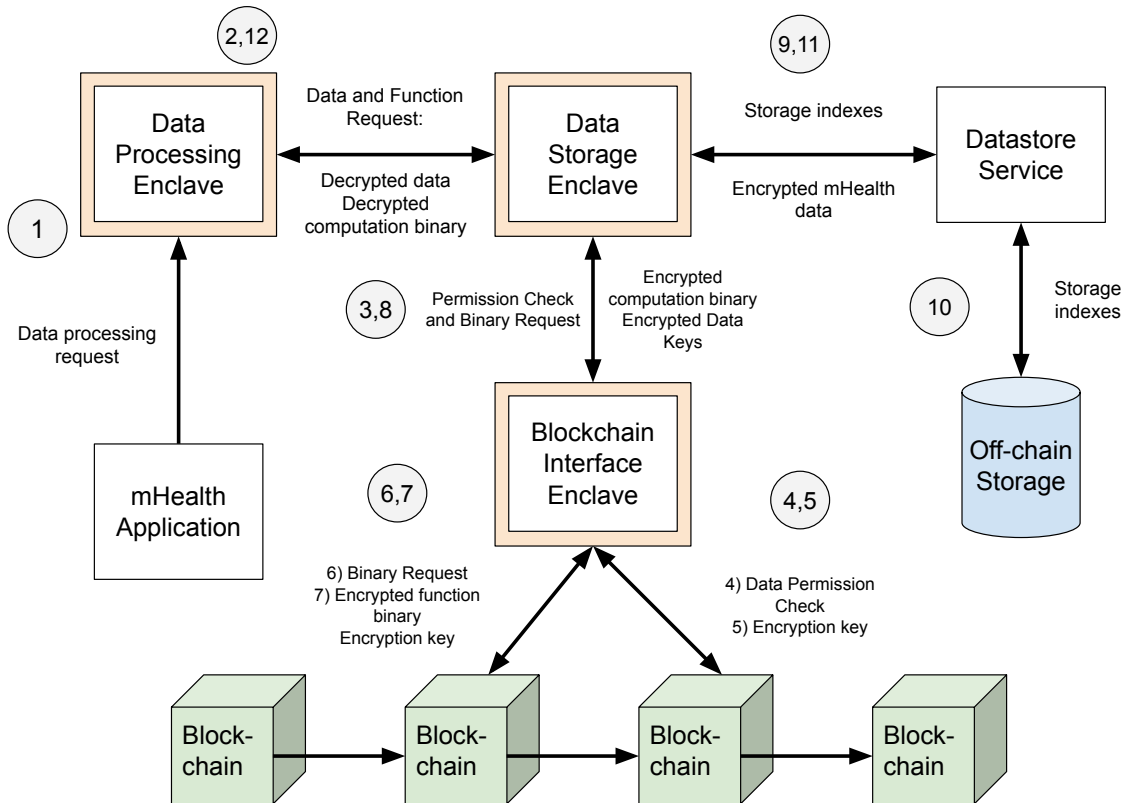


Figure 3.3: **Confidential Computations.** mHealth data stored within the Amanuensis system can be confidentially computed upon using a Data Processing Enclave. Before the confidential computation is performed, the provenance of the input mHealth data is verified to ensure the validity of any information resulting from the computation.

APP creates a signature, SIG_{APP} , over the request and appends it to the request to create message M_{APP} , which it sends to a Data Processing Enclave, DPE .

$$SIG_{APP} = SK_{APP}(PK_{APP} || PK_{CC} || L) \quad (3.9)$$

$$M_{APP} = PK_{APP} || PK_{CC} || L || SIG_{APP} \quad (3.10)$$

2. DPE requests the input data and confidential computation binary from a Data Storage Enclave, DSE , by forwarding it M_{APP} .
3. DSE requests provenance metadata and the confidential computation binary from a Blockchain Interface Enclave, BIE , by forwarding it M_{APP} .
4. BIE checks PK_{APP} against permission lists recorded in the blockchain for each of the specified input data sets contained in L .
5. For every input data set for which PK_{APP} has access, its provenance metadata is returned by the blockchain to BIE .
6. Similarly, BIE must also check that PK_{APP} has permission to use the computation specified by PK_{CC} .
7. If PK_{APP} has permission to use PK_{CC} , an encrypted binary containing the computation and an encrypted encryption key is returned by the blockchain to BIE .
8. BIE returns the provenance metadata and binary to DSE .
9. If the confidential computation binary or any provenance metadata is missing, or one or more permissions checks failed, DSE returns a permission failure to DPE . Otherwise, DSE verifies SIG_{APP} contained in M_{APP} .

$$(true|false) = PK_{APP}(SIG_{APP}) \quad (3.11)$$

If the permissions checks pass *and* SIG_{APP} is verified, DSE forwards L to the Datastore Service to request the input data sets.

10. The Datastore Service retrieves the encrypted data sets specified by L and returns them, C , to DSE .
11. DSE first decrypts each encrypted Data Key, J , contained in the provenance metadata packets using its Master Data Key, K_{MASTER} . Then, DSE uses each decrypted Data Key, K , to decrypt each encrypted health data set, C . With C decrypted, DSE may then verify the hash of the input data sets and their signatures. Finally, DSE verifies any enclave attestations contained within the provenance metadata packets by sending them to the attestation service. If all of these provenance checks pass, the plaintext data sets and confidential computation binary are returned to DPE .

$$M_{SP} = K_{HD}(C_{HD}) \quad (3.12)$$

$$where\ M_{SP} = D_{HD} || SIG_{HD} || SIG_{BOB} \quad (3.13)$$

$$(true|false) \leftarrow H_{HD} == h(M_{SP}) \quad (3.14)$$

$$(true|false) = PK_{HD}(SIG_{HD}) \quad (3.15)$$

$$(true|false) = PK_{SP}(SIG_{BOB}) \quad (3.16)$$

12. If DPE receives all of the specified input data and the confidential computation binary, then it can assume the application has the appropriate permissions and that the input data sets passed the provenance checks. It should then carry out the specified computation using the input data sets.

The results of the computation are then returned to APP , and written to off-chain storage by DSE (as described in the following section).

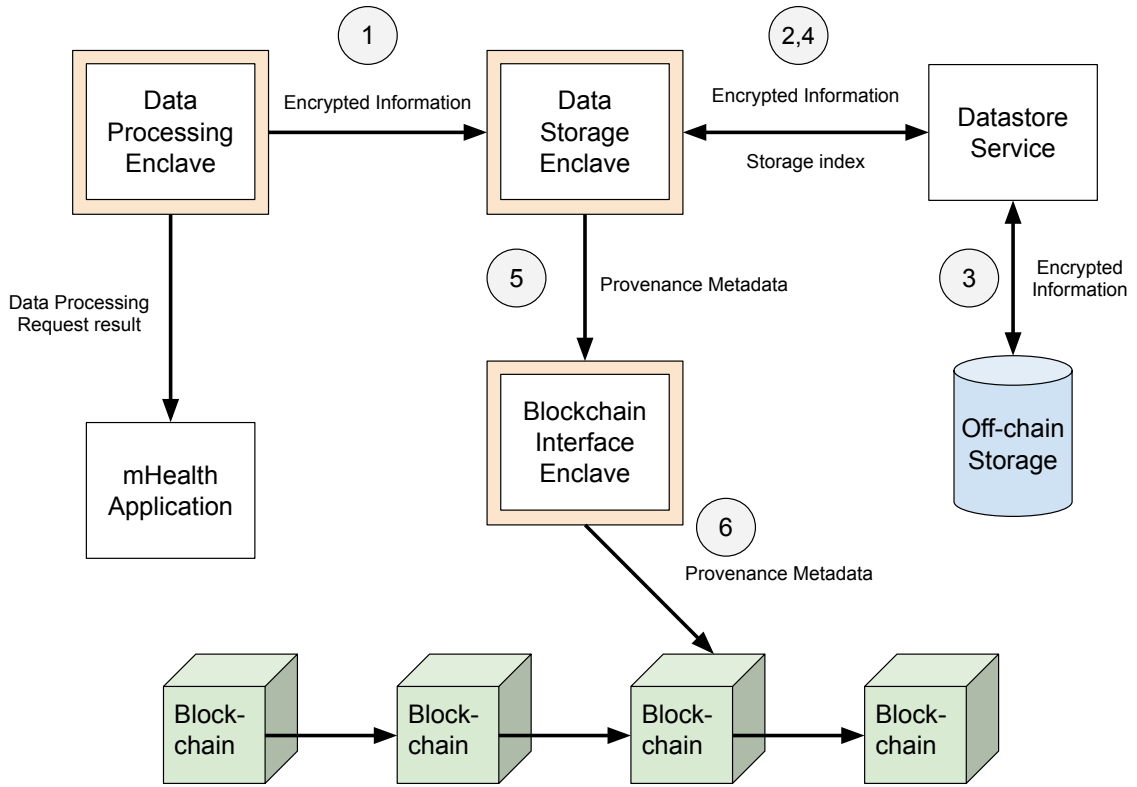


Figure 3.4: **Adding Information.** The output of confidential computations is new mHealth data, or *information*. Similar to the process for adding mHealth data, information is added to our system by a Data Storage Enclave. On top of the provenance metadata recorded for mHealth data, the Data Storage Enclave must record provenance metadata to show what computation was performed, and on what input data sets, to produce the resulting information.

3.8.3 Adding Information

Applications can process raw mHealth data and information via our Data Processing Service. Any new information created as a result of this processing must be added to the Amanuensis system and returned to the requesting application. This process is depicted in Figure 3.4 and is described in detail below.

1. After completing a data-processing request, a Data Processing Enclave, DPE , returns the resulting information, R_{CC} , to APP . DPE generates an attestation report, A_{DPE} , showing that it correctly executed the requested computation. This attestation report,

A_{DPE} , is then bundled together with the resulting information, R_{CC} , the application's Identification Key, PK_{APP} , the computation's Identification Key, PK_{CC} , and the list of input data set indexes, L , to form a message, M_{DPE} , which is sent to a Data Storage Enclave, DSE .

$$M_{DPE} = R_{CC} || L || PK_{APP} || PK_{CC} || A_{DPE} \quad (3.17)$$

2. DSE encrypts M_{DPE} with the Data Key, K_{CC} , which is unique to the computation that produced R_{CC} . The encrypted data, C_{CC} , is then sent to the Datastore Service for storage.

$$C_{CC} = K_{CC}(M_{DPE}) \quad (3.18)$$

3. The Datastore Service treats C_{CC} as a blob and writes it to off-chain storage.
4. The Datastore Service returns a storage index, I_{CC} , to DSE that indicates the location of C_{CC} .
5. After receiving I_{CC} , DSE generates an attestation report, A_{DSE} , and calculates a hash, H , of M_{DPE} . Then, DSE encrypts K_{CC} with the Master Data Key, K_{MASTER} , to produce J_{CC} . Finally, DSE bundles the provenance information into a packet, P_{CC} , and sends it to a Blockchain Interface Enclave, BIE .

$$H_{CC} = h(M_{DPE}) \quad (3.19)$$

$$J_{CC} = K_{MASTER}(K_{CC}) \quad (3.20)$$

$$P_{CC} = A_{DPE} || A_{DSE} || H_{CC} || I_{CC} || J_{CC} \quad (3.21)$$

$$(3.22)$$

6. BIE adds P_{CC} to the smart contract associated with CC .

3.9 Security Analysis

Drawing on the security model described in Section 3.3, and the system design described in Section 3.4, we can reason about Amanuensis’ ability to thwart the type of adversaries envisioned in our adversary model. Recall that physical and software attacks on mHealth apps and devices are out-of-scope because we assume that they operate correctly, can securely communicate with other system components, and can authenticate users (secure solutions to these challenges are well understood and out of scope of this thesis). Thus, here we need only consider an adversary that wishes to access health information via the blockchain, enclaves, or off-chain data storage. For reference, our adversary model affords the following capabilities to an adversary:

- An adversary can remotely compromise the TEE OS.
- An adversary can gain physical access to any TEE node.
- An adversary can gain control over (at most) a minority of nodes in the blockchain network.
- An adversary can arbitrarily read data from the blockchain and off-chain storage.

3.9.1 Blockchain Security

While the blockchain does not directly store mHealth data, it does maintain permission lists that determine who has access to, and may compute on, off-chain data. A blockchain adversary, A_{BC} , may attempt to modify these lists for the sake of tricking the rest of the system into thinking that A_{BC} has permission to consume some set(s) of mHealth data. Below we describe different attacks A_{BC} may use to modify permission lists, as well how Amanuensis prevents them from being successful.

Attack 1: Consensus Mechanism. Recall that a blockchain is a distributed-database that is updated via a consensus mechanism. When a node in the blockchain network wishes

to update the state of the blockchain, it broadcasts the changes to the rest of the network, which then performs a consensus protocol to determine whether the changes are valid. If the changes pass consensus, then the rest of the nodes update their local blockchain database to reflect the updated state of the blockchain. For the first attack, adversary A_{BC} tries to gain access to mHealth data by passing-off an illegitimate version of the blockchain to the rest of the network.

1. Assume that A_{BC} has gained control over a minority of the nodes in the blockchain network.
2. A_{BC} then modifies any permission list of its choosing by rewriting the local blockchain databases on the node(s) under its control.
3. After creating a false blockchain, B_{FALSE} , containing the modified permission lists, A_{BC} instructs its nodes to broadcast B_{FALSE} to the rest of the blockchain network.
4. At this point, the attack succeeds only if the rest of the network accepts B_{FALSE} as valid. The type of blockchain and consensus mechanism used in Amanuensis, however, requires a *majority* of the blockchain network to agree on changes to the state of B . Thus, since A_{BC} only controls a *minority* of the blockchain nodes, B_{FALSE} will be rejected by the consensus algorithm.⁴

Attack 2: Transaction Signatures. Transactions are the method by which users can interact with, and update, the state of the blockchain. For example, a data owner, Alice, can grant a data consumer, Bob, access to her mHealth data by submitting a transaction to the Amanuensis blockchain network stating that Bob should be added to her list of verified data consumers. In our second attack, A_{BC} will attempt to gain access to mHealth data by forging an illegitimate, but valid, transaction. In this attack scenario, let us assume that A_{BC}

⁴Note that exact numbers for what constitutes minorities and majorities will vary depending on the specific consensus mechanism used (e.g., minority for PBFT is $< 1/3$ but for PoW is $< 1/2$), but our assumption states that adversaries will never be capable of controlling a majority of the blockchain nodes in any blockchain implementation.

is attempting to gain access to Alice’s mHealth data by creating a transaction that appears to be from Alice and that grants A_{BC} access to her data.

1. A_{BC} may try to modify and replay one of Alice’s previous transactions so that instead of granting access to the intended consumer (say, Bob) it instead grants A_{BC} access to her mHealth data. This modified transaction, T_{FALSE} will not pass validation, however, as it contains a signature tied to the transaction data and by changing the data (i.e., changing Bob to A_{BC}) the signature will no longer match the transaction.
2. For A_{BC} to make the transaction appear valid, it would also need to modify the signature tied to the data within a transaction; thus, A_{BC} will need to gain access to Alice’s private signing key, SK_{ALICE} . A_{BC} could obtain SK_{ALICE} by attacking Alice’s smartphone where the key is stored, or by guessing the value of SK_{ALICE} .
3. Similarly, creating an entirely new transaction would require A_{BC} to gain access to SK_{ALICE} for generating a legitimate signature.
4. Given our security assumptions in Section 3.3, which states that all smartphones used to interact with Amanuensis have secure storage for encryption and signing keys, we know that A_{BC} will not be able to retrieve SK_{ALICE} from Alice’s smartphone. Similarly, given our cryptographic primitive assumptions, our attacker is incapable of guessing SK_{ALICE} in any practical time period.

3.9.2 Enclave Security

There are two enclaves that handle mHealth data in plaintext (Data Storage Enclave and Data Processing Enclave), and as such are prime targets for attack by some enclave attacker, A_{EN} . Below we describe different attacks A_{EN} may use to gain access to mHealth data being handled by the Data Storage and Data Processing Enclaves. Recall that side-channel attacks are out-of-scope for this thesis.

Attack 1: Data Processing Request. To ensure the confidentiality and provenance of mHealth data stored in Amanuensis, we require that computations on mHealth data take place within a Data Processing Enclave. To initiate a computation, data consumers must submit a data processing request detailing the specific input data and computation they wish to be executed. Much like the transaction signature attack on blockchain, A_{EN} may submit a false data processing request in an attempt to gain access to the result of some sensitive computation. In this attack scenario, let us assume that Alice has granted Bob access to her mHealth data, which A_{EN} will try to gain access to by forging a data processing request.

1. A_{EN} may try to modify and replay one of Bob's previous data processing requests so that instead of returning the result of the computation to Bob it returns to A_{EN} . In fact, A_{EN} may specify a different computation, one that simply returns the input data without performing any sort of computation over the data at all. This modified data processing request, R_{FALSE} will not pass validation, however, as it contains a signature tied to the request data and by changing the data (i.e., changing Bob to A_{EN}) the signature will no longer match the request.
2. For A_{EN} to make the request appear valid, it would also need to modify the signature tied to the data within a request; thus, A_{EN} will need to gain access to Bob's private signing key, SK_{BOB} . A_{EN} can obtain SK_{BOB} by attacking Bob's smartphone where the key is stored, or by guessing the value of SK_{BOB} .
3. Similarly, creating an entirely new request would require A_{EN} to gain access to SK_{BOB} for generating a legitimate signature.
4. Given our security assumptions in Section 3.3, which states that all smartphones used to interact with Amanuensis have secure storage for encryption and signing keys, we know that A_{EN} will not be able to retrieve SK_{BOB} from Bob's smartphone. Similarly, given our cryptographic primitive assumptions, our attacker is incapable of guessing SK_{BOB} in any practical time period.

Attack 2: Data Retrieval Request. One of the responsibilities of the Data Storage Enclave is to retrieve and decrypt mHealth data that the Data Processing Enclave needs for computations. A_{EN} may create a modified Data Processing Enclave that submits false data retrieval requests to the Data Storage Enclave. In this attack scenario, let us assume that A_{EN} has gained access to a TEE node so that it can deploy its modified Data Processing Enclave, DPE_{FALSE} .

1. Before enclave instances communicate with each other (e.g., send and receive data retrieval requests), they first perform an attestation process to ensure the authenticity and integrity of each other. This attestation process involves using a number of trusted-computing functionalities exposed to the enclaves by the Trusted Execution Environment in which they run. Thus, to submit a false data retrieval request, A_{EN} must create an enclave that passes the integrity measurement and attestation trusted-computing checks.
2. If A_{EN} were to create a custom DPE, DPE_{FALSE} , for submitting false data retrieval requests then the value of the integrity measurement would not match that of DPE_{TRUE} , as the measurements are calculating by taking cryptographic hashes over the enclave application. Thus, the probability of this attack being successful (i.e., A_{EN} creating a valid enclave image) is equal to the probability of A_{EN} finding two input values that map to the same output value of a cryptographic hash. Given our cryptographic primitive assumptions, our attacker is incapable of guessing a valid enclave image in any practical time period.

3.9.3 Off-Chain Data Storage Security

Lastly, an adversary A_{DS} may try to steal mHealth data directly from off-chain storage. We assume nothing about off-chain data-storage services other than they can be trusted to correctly write data to storage and return the correct data for a given storage index.

Attack 1: Reading Data. Given that off-chain storage is assumed to be insecure, and the storage indexes are stored on the blockchain for all to see, A_{DS} could simply retrieve data from off-chain storage. Since the data is encrypted by the Data Storage Enclave before transmission to a storage server, however, A_{DS} will not be able to decipher any of the information. To decrypt the mHealth data obtained from off-chain storage, A_{DS} would have to (1) gain access to the encryption key from the DSE’s secure memory or (2) guess the value of the encryption key. Both of these options contradict the security assumptions of our system, namely that A_{DS} cannot break into secure TEE storage and that A_{DS} is not capable of breaking cryptographic primitives (e.g., encryption algorithms) in a time period that is practically useful.

Attack 2: Modifying Data. A_{DS} may also attempt to *modify* data stored in off-chain storage. Indeed, because we assume these data-storage services are not secure, A_{DS} may succeed in adding fake data, replacing legitimate data with fake data, or deleting legitimate data. We consider data deletion to be a denial-of-service attack, however, which is out-of-scope for our work. Modifications to legitimate data, on the other hand, will be caught by the DSE the next time the data is requested, because the DSE performs integrity and provenance checks using metadata stored on the blockchain. To successfully modify mHealth data or information, A_{DS} would have to modify both the data in off-chain storage and convincingly update its corresponding metadata in the blockchain so as to pass the DSE provenance checks. Of course, to modify the state of the blockchain, A_{DS} would need to carry out an attack on the blockchain, which we have already shown to be improbable.

3.10 Implementation

Below we describe the specific blockchain and trusted execution environment technologies used in our prototype implementation of Amanuensis.

3.10.1 VeChain Thor Blockchain

We implemented the blockchain component of Amanuensis using the VeChain Thor platform [101]. Thor is based on the Ethereum [105] blockchain, and, as such, supports the account/balance model and provides a custom virtual machine for executing smart contracts. Thor differs from Ethereum in several ways, however, as it introduces a dual-coin model, a consensus mechanism called Proof-of-Authority (PoA), and support for transaction clauses and enforced transaction ordering. Unlike Proof-of-Work (PoW) consensus algorithms that allow anybody to validate transactions, PoA entrusts a small consortium of authorized entities (authority nodes) with validating and adding new blocks to the blockchain. Limiting those able to validate blocks makes PoA more centralized than traditional public consensus algorithms, but it also allows PoA to achieve much higher transaction throughput than its public counterparts. Amanuensis expects a consortium of healthcare-related service providers to manage the blockchain, not the general public, so Thor’s PoA consensus mechanism is a good fit for our system design. Furthermore, the higher transaction throughput is necessary as many mHealth data sources may frequently and periodically upload data to the system. Unlike Ethereum transactions, which are used to perform a single task (i.e., transfer crypto, call a smart-contract function), Thor transactions support “clauses” which make it possible to pack multiple tasks into one transaction and thereby increase the throughput of the blockchain platform. Our current implementation, however, does not make use of VeChain Thor’s transactions clauses and simply treats transactions as if they only have the capabilities of standard Ethereum transactions.

3.10.2 Intel SGX & GrapheneSGX

First introduced in their 6th generation processors (Skylake), Intel’s Software Guard Extensions (SGX) [50] has gone on to become one of the most widely used TEE platforms. As such, SGX hardware is readily available and its usage well documented, which is why we chose to implement the TEE component of Amanuensis with SGX. The challenge with

the SGX platform is that applications (i.e., Enclaves) need to be coded to explicitly use the SGX API to create, and execute in, a trusted context. We want users to upload their data processing programs to Amanuensis so that we can execute them inside of a TEE to provide data confidentiality and provenance, but users may be less likely to participate in the Amanuensis ecosystem if they need to modify their programs to fit the SGX paradigm. Thus, we use GrapheneSGX [31], a library OS that can be used to deploy unmodified applications on SGX. GrapheneSGX supports a number of unmodified applications including Apache, GCC, and the R interpreter [31].

3.11 Evaluation

In this section, we present an evaluation of our Amanuensis prototype. As the main contribution of this chapter is the architectural design of Amanuensis, the goal of the prototype implementation and evaluation is to demonstrate the feasibility of, and thus validate our decision to use, blockchain and trusted execution environment technologies for managing mHealth data. To this end, we evaluate the throughput of the blockchain for reading and writing data, as well as the overhead incurred by running applications on a trusted execution platform.

3.11.1 Blockchain Performance

Here we evaluate several important aspects of the Amanuensis blockchain: read latency, write throughput, and operational cost. Our blockchain tests were conducted against our *DataSource* smart contract, which we deployed on the VeChain test network.⁵ To deploy and interact with our smart contracts, we used VeChain’s ‘Sync’ web browser. Sync exposes an instance of their JavaScript library, Connex, which is used to create and submit transactions (e.g., deployments, smart-contract calls) on VeChain’s test network.

⁵Unfortunately, at this writing there is no public information available regarding the configuration of the VeChain test network.

Reading Data

Every time the Data Storage Enclave retrieves data from off-chain storage, it must also retrieve its associated provenance metadata recorded on the blockchain. In our implementation, there are two main functions related to retrieving provenance metadata: *getData()* and *hasDataAccess()*, where *getData()* returns the contents of our *record_t* struct and *hasDataAccess()* returns a boolean value indicating whether the specified consumer has permission to access the data. Transactions are not required when reading data from the blockchain, which means anyone can read from the blockchain without paying transaction fees (i.e., “gas”). We deployed our *DataSource* smart contract to VeChain’s test network and called each function 5,000 times. On average, *getData()* took 261 msec to return the provenance metadata and *hasDataAccess()* took 30 msec to return the boolean access value.

Writing Data

Every time the Data Storage Enclave writes data to off-chain storage, it also writes provenance metadata to the blockchain. Unlike reading data, writing data to the blockchain requires that the user (in this case, the BIE) submit a transaction to the miners. The VeChain Thor blockchain is theoretically capable of handling up to 10,000 transactions per second [73], but the maximum transaction rate seen on their production network thus far is 165 transactions per second [14].⁶ Regardless, to achieve practical transaction rates Amanuensis will require new data to be ‘batched’ at the app, device, or smartphone level, reducing the number of blockchain transactions required. Assuming 165 tx/s can be maintained, we calculated the number of mHealth data sources that the Amanuensis blockchain could support given they batch data every 1, 4, 8, 12, and 24 hours. The results of these calculations are presented in Table 3.3. Note that these calculations assume the blockchain is *only* being used for writing provenance metadata, and does not consider updating permissions lists, adding new smart contracts, or other actions that require blockchain transactions. Thus, the

⁶As of 3 March 2020.

values in Table 3.3 may overestimate the real number of mHealth data sources that could be supported.

Table 3.3: The number of mHealth sources Amanuensis blockchain could support for a given transaction rate and data batch.

Tx/s	Batch Rate (hours)	Sources
165	1	594,000
165	4	2,736,000
165	8	4,752,000
165	12	7,128,000
165	24	14,256,000

Gas Cost

In VeChain’s Thor (or other Ethereum-based blockchain systems), each blockchain transaction requires payment in a currency called *gas*. The gas concept, and the requirement to pay for transactions, was introduced by Ethereum as an incentive for miners to carry-out smart-contract invocations. In Thor, the fee size depends on the complexity of the smart-contract function. As an example of the cost for interacting with Amanuensis’ smart contracts, we present the gas costs for functions in the *DataSource* contract and their corresponding cost in U.S. dollars. We leave the decision of whether or not to require gas payments up to those implementing our system design, but we provide theoretical gas costs for their consideration. The VeChain Thor Token (VTHO) is used to pay gas costs on the blockchain, and 1 VTHO = 1000 gas. We can use the price of VTHO (\$0.0045585 [19])⁷ to estimate the dollar cost for interacting with the DataSource smart contract, as shown in Table 3.4. Depending on the batch rate of their mHealth data source, a user can expect to pay between \$0.47-\$11.28 per day to store their mHealth data in the Amanuensis system.

Although transactions on the VeChain Thor production network cost money, a hypothetical Amanuensis consortium can run its own blockchain and mutually agree to not charge any

⁷As of 13 April 2022.

Table 3.4: Monetary cost for each invocation of the DataSource smart contract.

Function	Cost (in Gas)	Cost (in \$)
Deploy Contract	711,385	\$3.24
addData	103,193	\$0.47
grantAccess	28,552	\$0.13
revokeAccess	25,639	\$0.12

fees for transacting on their blockchain instance, perhaps arranging off-chain mechanisms to share/distribute operational costs.

3.11.2 Enclave Performance

Here we present the impact of using Intel SGX (more specifically, GrapheneSGX) to execute sensitive mHealth data computations. Running applications on TEE platforms is inherently slower than running them on untrusted platforms, as the TEEs need to perform certain trusted computing features (e.g., data unsealing/sealing, integrity measurement, attestation) before, during, and after execution of an application. GrapheneSGX reports overheads ranging “from matching a Linux process to less than 2x in most single-process cases” [31]. Looking at their results, the operations that achieve overheads similar to that of an unmodified Linux process are those that do not involve memory allocation, garbage collection, or any other communications with the host system. Examples of operations that perform well in GrapheneSGX are matrix computations, sorting, and linear regression. We timed the execution for two programs running as regular Linux programs and as SGX enclaves within GrapheneSGX.⁸ Those programs are a sum of squares linear regression on one million data points and quicksort on a reversed array (worst case). The results are presented in Table 3.5, and looking at the “System” timings we can see that SGX requires a significant amount of time to initialize enclaves – on the order of seconds. This overhead is unfortunately unavoidable if we wish to execute in a confidential context. If we examine the time spent in “User” space, SGX imposes a 3.2x slowdown for quicksort and 1.9x slowdown for linear

⁸Tests were performed on a Dell Optiplex 7060 with Intel Core i7-8700 CPU running Ubuntu 18.04.4 LTS.

regression. Due to the enclave initialization and destruction time, computations on large datasets (i.e., those that will take a long time) will see less relative overhead imposed by SGX than short-lived computations.

Table 3.5: Execution times of plain Linux programs vs GrapheneSGX.

Benchmark	User	System
Quicksort	0.187s	0.000s
QuicksortSGX	0.591s	1.505s
Linear Regression	0.350s	0.004s
Linear RegressionSGX	0.661s	1.497s

3.12 Related Work

Amanuensis is certainly not the first blockchain-based health-data system nor the first TEE-enabled blockchain. Many prior systems provided inspiration and guidance in designing the Amanuensis system. One notable work is ShareHealth [33], a health-data system that secures mHealth data streams from various devices and vendors and provides cryptographic temporal-based access-control. Though ShareHealth does not use blockchain or trusted execution technology, it initially shaped the goals and design choices for Amanuensis. Below we detail prior research related to blockchain-based health-data systems and confidentiality-preserving blockchain techniques.

3.12.1 Blockchain-Based Health-Data Systems

Many have proposed that the transparency, immutability, and distributed trust afforded by blockchain may bolster the security, interoperability, and auditability of existing health-data systems [3, 7, 30, 75, 84, 88]. Based on recent surveys of the use of blockchain in healthcare [1, 37, 44, 46, 100], the majority of proposed blockchain-based health-data systems have focused on access control for provider-managed Electronic Health Record (EHR)

data [6, 16, 22, 23, 32, 47, 52, 67, 69, 74, 77, 104, 106]. These blockchain-based health-data systems allow patients greater control over their provider-managed health data by using a blockchain network to store and enforce data-access policies. Providers (and in some cases, patients) participate in running the blockchain network, thereby removing the need to trust any one entity with correctly upholding the data-access policies. Along with data-access policies, the blockchain stores hashes for maintaining the integrity of health data being stored off-chain and records instances of data access for auditing purposes. Though most of the surveyed systems focused on controlling access to EHR data, several leveraged the immutability and transparency of blockchain to increase the reliability of data provenance in applications like clinical drug trials, medical referrals, or risk and vulnerability assessment [52, 62, 63, 65, 70, 78, 80, 89, 110]. Others used blockchain to provide access control for mHealth data streams [18, 98, 99, 107].

While current blockchain-based health-data systems provide secure and distributed access control, data immutability, and auditability they do not support confidential and efficient ways for computing on sensitive health data. Smart contracts are capable of computation, but they would reveal sensitive health data to all of the network participants and be highly inefficient due to blockchain's transparency and the redundant execution of smart contracts by every mining node. It is important to note that blockchains can only attest to computations run *inside* the blockchain, but not actions that happen off-chain. Thus, while users can trust the computations that happen in the blockchain are valid and that the data is immutable, they have no way of verifying that the data added to the blockchain was created legitimately and that access policies are upheld outside of the blockchain.

3.12.2 Confidentiality Preserving Blockchains

Transparency is touted as one of the many benefits of blockchains, but it is also one of the main obstacles to blockchain's broader adoption. Certain applications only benefit from blockchain's distributed data-management model if said data remains confidential (e.g.,

collecting and evaluating bids for an auction). To this end, several techniques for achieving data confidentiality on blockchains have emerged: Zero-Knowledge Proofs (ZKPs), Multi-Party Computations (MPCs), and Trusted Execution Environments. ZKPs and MPCs provide greater cryptographic security guarantees than TEEs, but are computationally more expensive – making it difficult to support complex operations. Amanuensis uses TEEs for data confidentiality because we want to support general computations; we thus focus here on related work for TEEs.

Introduced by Ethereum, *smart contracts* allow application developers to deploy code to the blockchain network. These smart contracts serve as the back-end business logic component to front-end web applications, often referred to as decentralized applications (dApps). Several works address smart-contract data confidentiality by augmenting the blockchain with trusted execution environments so that sensitive smart contracts can be executed inside of a TEE instead of the mining nodes [13, 17, 54, 109]. Using this concept of a TEE-enabled blockchains, others use confidential smart contracts to build ML and AI data marketplaces [24, 25, 48], confidential cryptocurrency applications [8, 68, 97], and secure IoT [5, 26] and healthcare [66] data management systems. While these systems have the building blocks necessary to provide information provenance, only Liang et al. [66] explicitly attempts to do so. They take a passive approach to information provenance, however, and do not require periodic enclave attestations or verify data before it is used – data is only verified if requested, and only after the fact – whereas we perform regular attestations and verify the provenance of data before it is used.

3.13 Conclusion

Previous works on blockchain-based health-data systems largely focus on developing secure methods for sharing and tracking Electronic Medical Records (EMRs), but do not consider mHealth data. Furthermore, these systems only track the *movement* of health data, not

the *transformation* (e.g., computation, aggregation) of health data. In this chapter, we present a blockchain-based health-data system capable of securing mHealth data streams, providing individuals with a better view of how their data is being used, and allowing for the confidential, but verifiable, consumption of sensitive health data. By combining blockchain and trusted execution environments, Amanuensis provides participants with transparent and immutable data trails that can be used to verify mHealth data and information streams. Furthermore, Amanuensis allows users to confidentially share and consume said data and information. We present a prototype implementation of Amanuensis that is capable of supporting up to 14,256,000 mHealth data sources at \$0.47 per data source per day.

4

Provenance, Privacy, and Permission in TEE-Enabled Blockchain Data Systems

In previous chapters, we achieved information provenance by securing mHealth data at the point of its creation (Chapter 2) and at every point in which it is shared or transformed (Chapter 3). In this chapter, we build on our data-management system, Amanuensis, to address several issues that arise from the combination of blockchain and trusted execution environments. For instance, current TEE hardware platforms (e.g., Intel SGX [50]) have limited RAM (on the order of megabytes), whereas blockchain databases span tens to hundreds of gigabytes. This memory constraint makes running blockchain software

inside of a TEE impractical as the cost of reading blockchain data in/out of disk storage, decrypting/encrypting, and integrity checking may deteriorate blockchain performance to the point where it is no longer useful to higher-level applications. Thus, programs running *inside* of TEEs must rely on software *outside* of the trusted area to retrieve data (e.g., a data analysis program and encrypted sensitive inputs) from the blockchain database. Furthermore, while we have removed sensitive data operations from the blockchain (i.e., data computation), simply tracking *who* is interacting (and with *what*) on the blockchain may reveal sensitive information about an individual or organization (e.g., Bob is sharing data with an oncologist).

4.1 Contributions

We build on our TEE-enabled blockchain data system, Amanuensis [38], which is designed to facilitate the sharing and computation of mobile health (mHealth) data between disparate healthcare providers. Namely, this chapter makes three major contributions:

- assured freshness of access-control lists stored on the blockchain,
- expanded privacy for users interacting on blockchain, and
- secured protocol for verifying the provenance of data produced by confidential TEE programs.

We expand and evaluate our Amanuensis prototype to measure the cost of information provenance and confidentiality. Although the impact of our solution is substantial at times – with slowdowns ranging from 3.2x to 188x depending on program and input configuration – applications that require confidential but verifiable computations may find these overheads acceptable (e.g., the offline processing of sensitive health data). Though our work is framed in the context of mHealth data, the techniques and protocols within can be applied to sensitive data of any nature.

4.2 Background

We expand on the blockchain and TEE background information provided in Section 3.2 by describing several concepts related to our data freshness, identity privacy, and information provenance solutions.

4.2.1 Blockchain

Blockchains that support smart contracts are implemented with two databases. The first is an append-only database of time-stamped blocks that are cryptographically linked to protect them from tampering and revision. The second is a mutable database that stores the current state of smart contract variables and memory. The integrity of the smart contract data is maintained via **Merkle tree**, which is a binary tree built by hashing the leaf node values (e.g., smart-contract state) and iteratively hashing pairs of hashes together until one value is left. This final hash value essentially serves as a ‘snapshot’ of the data in the leaf nodes and is re-calculated and stored within every new block that gets added to the first database. Given the newest block in the blockchain, a user can validate smart contract data by recreating the Merkle tree path from the leaf node (i.e., smart contract data) to the root node (i.e., integrity snapshot). This path is known as a ‘Merkle proof’ and consists of $\log_2(N)$ values, where N is the total number of smart contracts on the blockchain [15].

Users submit **transactions** to the blockchain network that contain directives for updating the state of the blockchain (e.g., record a new piece of data, invoke a smart-contract function). Contained within these transactions are the identities of the users that authorized said transactions. Instead of identifying users by their name, however, they are identified via cryptographic public-keys (represented as alphanumeric character strings). For this reason, blockchains are considered to be ‘pseudo-anonymous’, but this level of anonymity has been shown to be easily broken [41, 79, 85].

4.2.2 Trusted Execution Environments

The Intel SGX Trusted Execution Environment provides Enclaves with a trusted-computing feature known as **remote attestation**, which is a process for proving to parties outside of the TEE platform (i.e., code executing on a different machine) that an Enclave has been unmodified and is running on genuine Intel SGX hardware. The first step in remote attestation is **local attestation**, which is the process where an attesting Enclave proves to the **Quoting Enclave** – a trusted program written by Intel – that it is running in a TEE context on the same hardware platform. Specifically, the attesting Enclave invokes special hardware instructions to take cryptographic integrity measurements of its code and data. These measurements are bundled and signed with a key only available to programs running in the trusted execution context (a key that is baked into the hardware at manufacture time). This signed message is then sent to the Quoting Enclave where it is verified, signed, and returned to the attesting Enclave. This second message, or ‘quote’, is then sent to the Intel Attestation Service (IAS) where it is verified, signed, and returned to the attesting Enclave. This last message is the **attestation report** and it contains the quote verification status, the quote, other platform related information, and an IAS signature over the report. Finally, the attesting Enclave provides the remote verifier (i.e., the program that asked the Enclave to attest to itself) with the attestation report, which can be used to verify the integrity and authenticity of the attesting Enclave along with the fact that it is running on genuine Intel SGX hardware.

4.2.3 Amanuensis

Amanuensis is a TEE-enabled blockchain system designed to facilitate the sharing and computation of mobile health (mHealth) data between disparate healthcare providers [38]. The backbone of Amanuensis is its consortium-based blockchain, which provides a suite of smart contracts that allow individuals to share and manage their mHealth data in a confidential yet verifiable fashion. The consortium is comprised of health-related organizations

(e.g., hospitals, clinics, insurance providers, public-health agencies, researchers, government regulators) that collectively manage the blockchain. This approach eases inter-organization information sharing, because they can use the blockchain to verify the authenticity and validity of shared data. As a consortium blockchain, permission to execute, validate, and add transactions to the blockchain is reserved for miner nodes operated by the consortium members, but, as the contents of the blockchain are public, individuals can still read data from the blockchain and submit transactions they wish to be executed. Without a solution like Amanuensis, individuals must trust health-service providers to manage their private health data – and largely do – but they have no way to *verify* that their data is being handled correctly. By forcing healthcare providers to use a publicly-readable blockchain for data-access decisions, individuals (or watchdogs like consumer-protection organizations) are afforded a look into how sensitive data is managed. The exact number of nodes required to run blockchain component of Amanuensis is implementation-specific, and thus left to the members of the health consortium; the number of nodes depends on many factors, including the number of participants in the network, availability requirements, and cost. All our design for Amanuensis requires is that each consortium member participates in the blockchain network in some capacity so as to ensure the availability of the network and to maintain the distributed and immutable nature of the blockchain.

Amanuensis also maintains a distributed network of **TEE servers** to ensure the availability of trusted applications (SGX Enclaves), which are used by individuals and organizations to share and compute upon their health data. As with the blockchain mining nodes, TEE nodes are maintained by members of the blockchain consortium (not by individuals). Amanuensis requires all actions upon mHealth data to occur inside a TEE to ensure that data remains confidential and that actions can be verified. Utilizing the TEE’s ‘remote attestation’ feature, enclaves can prove (to relevant individuals and organizations) that they are legitimate Amanuensis Enclaves executing on genuine TEE hardware. Attestation reports produced by enclaves, along with other relevant metadata, are stored on the blockchain

to create a transparent record of data creation, storage, sharing, and modification that can be verified by data owners and consumers without the need to provide them access to the source data. While provenance metadata is stored in the blockchain, the actual health data is encrypted and stored in off-chain storage services. **Storage services** may be owned and managed by different service providers, but the keys used to encrypt health data reside in protected TEE memory and on the blockchain (in encrypted form). Thus, once stored in the system, mHealth data can only be decrypted by one of our secure enclaves running in our TEE network. Again, like the blockchain component, the number of TEE nodes is implementation-specific and left to the members of the health consortium. Ideally, they would choose a number that is sufficient to guarantee availability of the system and to ensure that the TEE component remains distributed across the consortium members. In other words, there should not be a single, centralized, TEE node or cluster that is responsible for granting access to the blockchain network and off-chain datastore.

4.3 Security Model

Amanuensis aims to preserve the confidentiality, integrity, and provenance of sensitive health data. More specifically, data confidentiality, integrity, and provenance must be upheld for both source data *and* derived information (i.e., information created through a series of aggregations and transformations of source data). Below we present the capabilities afforded to an adversary and the security assumptions for each of the technological components of Amanuensis.

4.3.1 Adversary Model

We consider an adversary that (1) wishes to gain access to private health information to learn something sensitive about one or more individual(s), (2) wishes to modify private health information, or (3) wishes to pass off false health information as authentic. We assume

the adversary can (1) gain physical access to a TEE node or remotely compromise the OS, (2) gain control over some small¹ portion of the blockchain network, and (3) arbitrarily read data from the blockchain and off-chain data storage.

4.3.2 Security Assumptions

While some of the following security assumptions may seem implicit based on the technological definitions presented in Section 3.2 and Section 4.2, we choose to explicitly state them to clearly define the scope of security challenges that our system aims to address. Furthermore, we use these assumptions as the basis for our security analysis in Section 3.9. The security of any system depends on both its design *and* implementation, but the focus of this paper is evaluating the design of our system. Hence, security challenges arising from implementation choices (e.g., side-channel attacks on Intel SGX) are largely left out-of-scope.

Blockchain Amanuensis uses a blockchain to maintain the integrity of the data it stores and of the smart-contract operations it performs. For these properties to hold, we must assume that the consensus algorithm is capable of preventing attackers from modifying the blockchain even if they control some miners in the blockchain network. That is, we assume that the majority of the miner nodes act in good faith when executing the consensus protocol.

Trusted Execution Environments Amanuensis relies on TEEs to provide a secure and confidential computing environment for handling sensitive mHealth data. We assume that the TEE hardware and firmware are designed such that they guarantee the authenticity of the executed code, the integrity of the run-time states, and the confidentiality of the Enclave’s code and data – while preventing outside entities (OS, hypervisor, applications) from gaining access or control. Furthermore, we assume that the TEE hardware has the

¹Blockchain networks that use Proof-of-Work (PoW) consensus algorithms can operate correctly if the majority of the mining nodes act in good faith.

capabilities for correctly and securely providing the trusted computing features described in Section 3.2 (e.g., integrity measurement, remote attestation). We also consider side-channel² and Denial-of-Service (DoS) attacks out-of-scope.

mHealth Apps and Devices We assume that mHealth apps and devices, regardless of type or manufacturer, operate correctly; they produce accurate data and they can securely communicate with a smartphone. We also assume that they have ways to authenticate the user of the device so as not to label data with the wrong data subject. Physical attacks on mHealth apps and devices are out-of-scope.

Smartphone We assume that an individual’s smartphone correctly and securely aggregates data from the individual’s mHealth apps and devices and sends the data batches to one of the TEEs for storage. In this context, the smartphone correctly executes our health-data-collection application and has secure storage for protecting encryption and signing keys and sensitive health data.

Off-Chain Data Storage We assume nothing about off-chain storage services other than they can be trusted to correctly write data blobs to storage and return the correct data blob for a given storage index.

Trusted Time Service We assume that the time service can be trusted to respond with the correct wall-clock time upon request, and that communications between the time service and enclaves are secure.

Cryptography We assume that the underlying cryptographic primitives on which our system is built (e.g., cryptographic hashes, digital signatures, encryption algorithms) are not

²In concept, TEEs protect against various forms of physical attack, but in reality some implementations have been shown to be vulnerable to side-channel attacks, which leverage observable physical traits (e.g., memory-access patterns, CPU power-draw) of a computing system to infer what operations it is performing and/or what data it is handling.

breakable in any practical time frame.

4.4 Data Freshness

The ‘transparency’ and ‘decentralized trust’ that blockchain purportedly offers is only truly available to those who participate in maintaining and updating the state of the blockchain (i.e., miners). By keeping their own local copy of the database and participating in consensus, miners are assured that the state of their blockchain (and thus the data contained within) is correct.³ Other participants in the blockchain network, however, are required to place a certain degree of trust in miners. Indeed, a user that queries a miner for blockchain data *trusts* that the returned data is correct but there remains a possibility that the data is wrong (either through malicious actions or some error on the miner’s part). To verify the received data, a user would have to contact multiple miners to request the same piece of data (which still only provides some statistical guarantee of correctness) or they would have to download the entire blockchain and validate all transactions starting from the genesis block up to the most current block.

Current Intel SGX implementations have limited RAM availability for enclaves (128-256 megabytes), but blockchain databases tend to span tens to hundreds of gigabytes. The cost of paging data in and out of disk for enclaves is particularly expensive, because they must perform encryption and integrity checks on all data that pass through the trust boundary. This high paging cost makes running a miner inside of an enclave impractical for most applications and is the reason Amanuensis uses enclaves for performing sensitive data operations but not for maintaining the blockchain. The issue with this split computation paradigm, however, is that the enclave is not able to validate data received from the blockchain. When an enclave receives a request for a sensitive data operation it first queries the blockchain for the access-control list to check that the requester has the proper permissions. How then

³Data entered into the blockchain is subject to change for some period of time; users must wait until the data is sufficiently ‘deep’ in the blockchain to be assured that the state will not change due to a fork [45]

does the enclave know that the list returned to it is correct and that it is the most recent version? Without having its own copy of the blockchain, the enclave has no way to prevent a malicious miner from feeding it a false or stale access-control list.

Below we detail how to employ the digital certificate validation method known as NOVOMODO in conjunction with a trusted timing server to verify the freshness of access-control lists received from the blockchain.

4.4.1 NOVOMODO

Amanuensis maintains a smart-contract instance on the blockchain for each mHealth data source, and each instance contains a list of approved data consumers for that mHealth data source. In our prior design, these lists contained the blockchain IDs of the approved data consumers, but we now propose tracking digital certificates instead – certificates which we will verify using NOVOMODO: a scalable small-bandwidth method for validating the integrity, authenticity, and freshness of digital certificates. The certificates contain the following information:

- **blockchain address** – address of the data consumer’s blockchain account
- **issue date** – date and time that this certificate was issued
- **expiration date** – date and time that this certificate expires
- **revocation period** – granularity of time after which revocation takes effect (e.g., one day)
- X_n – the ‘validity target’, or value used to prove freshness of this certificate
- $\text{Sig}_{\text{OWNER}}$ – signature of the data source owner

To explain how validating a NOVOMODO certificate works, we must first look at how X_n is generated. Imagine an mHealth data source owner, Bob, who wishes to grant a data

consumer, Alice, access to one of his mHealth data sources for a period of one year with a revocation period of one day beginning on 1 January.

1. Bob generates a random 20-byte value known as X_0 .
2. Bob hashes X_0 $n = 365$ times to produce values X_1 through X_{365} , where $X_i = H(X_{i-1})$ and $H(x)$ is a cryptographically secure one-way hash function.
3. Bob makes the final value X_{365} (i.e., the validity target) public by including it in the signed certificate, but keeps X_1 through X_{364} secret.
4. As the year passes and as long as Bob desires Alice to retain access, Bob sends Alice a ‘validity proof’, X_{365-i} , on day i .
5. Suppose Alice desires access to the data on day i ; Alice sends an enclave a data request containing the validity proof X_{365-i} .
6. The enclave verifies that Alice still has access to Bob’s data by hashing X_{365-i} i times; if the final hash value matches the validity target X_{365} contained in the certificate then Alice must still have permission to use Bob’s data.

Unless Alice was (1) somehow able to steal X_{365-i} from Bob’s secure storage or (2) break the cryptographic hash function used to produce X_{365} , we can assume X_{365-i} proves the continued validity (i.e., freshness) of Alice’s data access permission. Bob can suspend or revoke Alice’s access simply by not providing her with that day’s validity proof. The effect of revocation may be delayed as long as the revocation period – if Bob already provided Alice with that day’s validity proof then the certificate will be valid for the remainder of the day even if Bob removes the certificate from the blockchain (assuming a malicious miner saves and provides the certificate to an enclave even after it was removed).

The NOVOMODO method alone is not enough to guarantee freshness of the certificate, however, as it relies on the enclave having access to a trusted time source. Indeed, the

enclave is in the same situation as before: it has no way to determine whether the X_{365-i} value provided by the data requester is for the current or past revocation period.

4.4.2 Freshness Protocol

SGX enclaves do not have a trustworthy real-time clock, but we can assume the existence of a trustworthy time service, available over the Internet, whose reports of the current real time can be cryptographically verified as authentic.⁴ The challenge, however, is for an enclave to ensure that any such reports are ‘fresh’, that is, to bound the amount by which the reported time lags behind the true current time. Recall that enclaves can only attest to information *inside* of the TEE boundary; any data received from outside of this boundary (e.g., network messages) are subject to modification or delay by the host operating system (OS) [4].

To illustrate why this delay is problematic, imagine that Bob decides on the afternoon of May 11 to revoke Alice’s access to his data for the remainder of the year and tells her as much. Furthermore, assume that Bob had already sent her May 11’s validity proof, X_j . By NOVOMODO’s definition of revocation, it is valid to allow Alice access to Bob’s data until the end of May 11, but not after.

Alice, not wishing to respect this constraint, may conspire with a malicious OS on an enclave server to delay responses to that enclave from the time service – responses that contain a wall-clock time falling on May 11 – such that the responses are delivered to the enclave *after* May 11. If, for example, the enclave asks the time service for the time, on May 11, and the service returns a timestamp like 8:01pm May 11, the malicious OS might not deliver that message to the enclave until May 12 or even later. The enclave would then believe that the validity proof X_j is still fresh and thus that Alice currently deserves access to Bob’s data, violating the revocation intent. Alice’s requests to that enclave server may thus provide her access to data she is not authorized to receive.

⁴The Network Time Security (NTS) protocol for example, which secures the Network Time Protocol (NTP) via TLS [34].

Fortunately, there is a straightforward solution. Every computation request requires four fundamental steps: (1) verify the requester’s authorization to access the input data, (2) retrieve the input data, (3) perform the computation, (4) write the output data. The first step requires the enclave to contact the time server for a certified statement about the current time; on receipt, though, it only knows that the actual time is *later than* the reported time. If the actual time is much later than the reported time, the second step might access data for which the requester is no longer authorized. The solution is to switch the order of steps (1) and (2): to fetch all the input data, then request the current time from the time server and verify it against X_j ; if valid, the enclave can be assured the access occurred at a valid time because it occurred before the timestamp was even requested from the time server. If not valid, the data is discarded and the computation cancelled.⁵

4.5 Identity Privacy

As mentioned in Section 4.2, users are identified on the blockchain not with their real name but with a cryptographic public-key (represented as an alphanumeric character string). The use of these strings provides a level of ‘pseudoanonymity’ that (unfortunately) in many applications is easily broken. For instance, imagine an individual Bob who wishes to share some mHealth data with his healthcare provider Alice. Given the nature of the healthcare provider-patient relationship, Alice almost certainly knows Bob’s real-world identity. When Bob grants Alice access to this data by updating the access-control list on the blockchain she will learn his blockchain pseudonym as well. By searching the blockchain for all instances of that pseudonym, Alice can determine if Bob has relationships with other healthcare providers. So, even though the blockchain is pseudonymous, it may still leak private information about its users.

⁵Although this approach may waste some effort, fetching data that may no longer be allowed for use, it would only happen when access has been revoked and the time-server response has been inordinately delayed; an unusual occurrence.

The issue of **identity privacy** – a blockchain user’s ability to conceal personally identifying information (PII) from other users of the blockchain – has not gone unnoticed in the blockchain community. There exist several solutions to this problem (e.g., zero-knowledge proofs, ring signatures, mixing services), none of which we believe are a good fit for our system design. Coincidentally, by forcing Amanuensis users to submit data computation requests directly to enclave instances, we already have a partial solution to identity privacy. Indeed, the only blockchain related by-products of these requests are *enclave generated* transactions containing provenance metadata for validating the computation output. Anybody with access to the blockchain can track and validate the creation of new data (i.e., information), but they cannot discern who is responsible for the request as the only identity tied to the transaction is the enclave’s. We take this idea, which we refer to as **enclave proxies**, and extend it to every transaction in the Amanuensis blockchain network (instead of just provenance metadata transactions).

4.5.1 Enclave Proxies

Our enclave proxy model requires Amanuensis users to submit all blockchain transactions to enclave instances instead of directly to the blockchain network. Enclaves then rewrite transactions to make it appear as though the enclave was the creator of the transaction before submitting them to the blockchain network on behalf of the end user. Before we describe the necessary changes to transactions and smart contracts, we define several relevant terms.

DataSource Smart Contract

records information for identifying a particular mHealth data source (e.g., mHealth application or device), the data that it produces, and data-access policies. On the blockchain, there is an instance of the DataSource contract for every mHealth data source in the Amanuensis system.

Data Key

a symmetric encryption key that is used to encrypt data streams. Data keys are used by the Amanuensis enclaves to encrypt incoming mHealth data streams before writing them to off-chain storage. We store Data Keys on the blockchain so that they persist and are accessible to all enclave instances; when necessary, any TEE server may service the mHealth data source. Data keys are first encrypted with the Master Data Key before being written to the blockchain.

Master Data Key

the symmetric encryption key that is used to encrypt and decrypt Data Keys stored on the blockchain. Whereas there are multiple Data keys, there is only one Master Data Key in the Amanuensis System. During a one-time setup phase, the Master Data Key is generated by one enclave, which then registers itself with the *EnclaveRegistry* contract. Subsequent enclaves that join the network can look-up existing enclaves on the registry and retrieve the Master Data Key through the remote attestation and secrets provisioning mechanisms provided by TEEs.

4.5.2 Smart Contracts

By removing user pseudonyms from transactions, we eliminate an attacker's ability to trace a user's activity on the blockchain. A by product of our proxy method, however, is that we have undermined the blockchain network's ability to verify (i.e., authenticate) that a given user has the authority to make a particular transaction (e.g., update an access-control list in a DataSource contract). Indeed, smart contracts often include code at the beginning of sensitive functions to check that the caller is the owner of said smart contract. So, how do smart contracts authenticate users hidden behind enclave proxies? Instead of relying on smart contracts to verify function callers, we require that enclaves perform this verification

before making any call to a smart contract.

This authentication process is detailed below, beginning with the deployment of a smart contract:

1. Bob obtains a new mHealth device whose data he wishes to manage with Amanuensis. Bob creates a transaction containing a contract creation call for one of our *DataSource* smart contracts, which he then sends to an enclave instance.
2. Upon receiving Bob's contract creation request, the enclave rewrites the transaction so that the enclave appears as both the creator of the transaction and *owner of Bob's new smart contract*. The enclave then generates a new *Data Key*, which it uses to encrypt Bob's public blockchain pseudonym. The *Data Key* is then encrypted with the *Master Data Key*. Both Bob's encrypted pseudonym and encrypted *Data Key* are included as an argument to the smart-contract creation call. The newly formed (and signed) transaction is then submitted to the blockchain network.
3. Afterwards, the enclave returns the transaction receipt received from the blockchain network to Bob, which he can use to locate his smart contract.
4. With Bob's smart contract successfully deployed and containing his encrypted pseudonym, the enclave need only retrieve and decrypt the pseudonym to verify subsequent calls to his smart contract.

4.5.3 Transactions

The process for rewriting transactions is as follows:

1. A user, Bob, wants to grant Alice access to the data produced by one of his mHealth devices. Bob creates a blockchain transaction containing a call to the *grantDataAccess* function of the smart contract associated with the particular mHealth device that he

wishes to share. Bob signs the transaction with his private key (the one associated with his public blockchain pseudonym) and sends this transaction to an enclave instance.

2. Upon receiving Bob's transaction, the enclave first verifies Bob's signature by retrieving his public identity, which is stored in an encrypted form in his mHealth device's smart contract (the details of which are described in Section 4.5.2).⁶
3. If Bob's signature is verified, the enclave generates a transaction identical in content but signed by the enclave. The newly created and signed transaction is then submitted to the blockchain network.
4. The enclave returns the transaction receipt received from the blockchain network to Bob so that he may locate the block containing his transaction.

It is important to note the significance of encrypting Bob's public pseudonym before storing it on the blockchain with his mHealth device's smart contract. Since Data Keys are unique to each smart contract (and thus to each mHealth data stream), Bob's encrypted pseudonym will also be distinct across every smart contract that he owns. In other words, we have effectively provided Bob with multiple pseudonyms that prevent attackers from linking his smart contracts with each other. Thus, even if Alice somehow learns Bob's encrypted pseudonym for a particular smart contract she cannot use it to locate his other smart contracts. Similarly, by encrypting the pseudonyms of authorized data consumers (e.g., Alice) with the Data key we can prevent the linking of data consumers across mHealth devices and patients. Even though Bob knows Alice's encrypted pseudonym for his particular smart contract, he cannot use that to locate other smart contracts that she has been given access to as her encrypted pseudonym will be different.

⁶Normally, Bob's signature would be verified by miners in the blockchain network, but in our new protocol miners never see Bob's signature as it is replaced by the enclave's. Thus, it is now the enclave's responsibility to ensure that Bob has the authority to make that particular smart-contract call.

4.6 Information Provenance

One of the goals of Amanuensis is to provide *information provenance* for data generated via one or more computations on intermediate and source data sets. The key to our information provenance solution is the remote attestation feature provided by our trusted execution environment. Previous systems (including our own) claim that by having an enclave generate an attestation we are guaranteed the program’s output to be ‘correct’ (i.e., that the program’s code is unchanged and thus that any output it generates is as expected). Furthermore, that we can use attestations to prove to other parties that the enclave’s output is correct. How exactly does this attestation process work, and how does it fit into the overall system to achieve a goal such as information provenance?

It turns out providing information provenance is more nuanced than simply “requesting an attestation” and passing it on to the next data consumer. While attestations are cryptographically tied to an enclave’s code and the hardware platform that it is running on, they are not inherently tied to the program’s output. For example, imagine that Bob connects to an Amanuensis enclave and asks for a remotely-verifiable attestation report. The enclave returns a report to Bob, which he uses to verify that (1) the enclave is running on genuine Intel SGX hardware, (2) that its code has not been modified, and (3) that it is the authentic Amanuensis program it purports to be. Satisfied, Bob requests that the enclave perform a computation on some mHealth data. The enclave performs this request and returns the output of the computation: 42. Having just requested and verified an attestation report, Bob accepts that 42 is the correct output of his requested computation. Now say Bob shares this result and attestation report with Alice. Is she assured that 42 came from Bob’s specific computation request? Alice is able to verify that the report was generated by a genuine Amanuensis enclave running on Intel SGX hardware, but there is nothing to tie the report to the output 42 – Bob could have provided her with any number and she would have no way to determine how and when it was generated by the enclave.

In the following subsections we detail our method for achieving information provenance

via attestation reports and metadata recorded on the blockchain.

4.6.1 Provenance Packet Structure

For every data upload and computation, an enclave generates a corresponding ‘provenance packet’ and uploads it to the blockchain. This packet includes the following fields:

- **storage index** – index for locating the output data set
- **input count** – number of input data sets to the computation (0 for data upload)
- **input indexes (optional)** – indexes for locating input data sets to computation
- **input hashes (optional)** – cryptographic hashes of input data sets to check integrity
- **output hash** – cryptographic hash of output data set to check integrity
- **computation nonce (optional)** – nonce to distinguish between identical requests
- **computation ID (optional)** – unique string identifying the computation performed

The same packet structure is used for both data upload and computation requests, though some fields are left empty for data uploads. The importance of this packet is that it can later be used (in combination with an attestation report) to verify a specific action taken, and specific output generated, by an Amanuensis enclave.

4.6.2 Attestation Reports

Intel SGX attestation reports contain (1) a report signature, (2) an embedded ‘quote’, and (3) embedded ‘user data’. Generally speaking, attestation begins with an enclave generating a ‘quote’, which is then sent to the Intel Attestation Service (IAS) for verification. It is the IAS that responds with an attestation report, not the enclave, at which point the remote verifier performs the following checks.

1. Verify the report signature for authenticity; if the signature was made by IAS, then the verifier can trust the rest of the information embedded within the report.
2. Check the quote status; if OK, then IAS has determined the provided quote was generated by genuine Intel hardware.
3. Check the enclave signature within the quote to ensure that it is a genuine Amanuensis enclave.
4. Check the enclave measurement value to ensure that the enclave's code has not been modified.

If all of the checks pass, then the remote verifier is assured to be communicating with an unmodified Amanuensis enclave running on genuine Intel SGX hardware. Not yet mentioned is 'user data', which is an optional 64-byte field containing user-defined data provided at the time the quote was generated. It is this field that enables us to tie the enclave to its specific actions and outputs. After completing a request, an enclave calculates a cryptographic hash of the output provenance packet and includes it as the 'user data' in the attestation quote. Upon receiving the attestation report from the IAS, the enclave uploads both the provenance packet and attestation report to the blockchain, which together contain all of the information needed to verify the provenance of the enclave's output.

Now, for every data set uploaded to Amanuensis there is a corresponding attestation report *proving* that an enclave received and saved said data. Furthermore, before a data set is used in a computation, an enclave fetches and verifies its attestation report and provenance packet to ensure the authenticity and integrity of the input data set. Afterwards, the enclave generates an attestation report for the output data, which is tied to the input report(s) by the 'user data' field. Thus, we have a chain of reports and provenance packets that can be used to verify the provenance of data that has been created through a series of aggregations and transformations (i.e., information).

4.7 Security Analysis

Based on the security model described in Section 4.3, and our data freshness (Section 4.4) and identity privacy (Section 4.5) solutions, we can reason about Amanuensis’ ability to thwart the type of adversaries envisioned in our adversary model. For reference, our adversary model and security assumptions allow an adversary to

- compromise the TEE OS (locally or remotely),
- compromise some nodes in the blockchain network, and
- read data from the blockchain and off-chain storage,

but do not allow an adversary to

- compromise TEE hardware protections,
- control a majority of nodes in the blockchain network,
- compromise mHealth apps and devices,
- compromise the trusted time service, or
- break standard cryptographic algorithms and protocols.

These latter assumptions are foundational to secure systems, or have well-known solutions outside the scope of this paper. Given this security model, we present the possible attack scenarios on data freshness and identity privacy (and how our system design prevents them).

4.7.1 Data Freshness

An adversary that previously had legitimate access to some data set may attempt to re-gain access by passing off a stale data-access certificate to one of the Amanuensis TEEs. Thus, our system must ensure that attackers are not able to (1) manipulate the TEE’s notion of time

to pass off stale certificates as valid (i.e., ‘fresh’) or (2) gain access to the current revocation period’s validity proof.

Attack 1: Time Manipulation. Recall that data access is granted by a data owner generating and signing a NOVOMODO certificate containing the data consumer’s identity and a validity target. For each revocation period, the data owner provides the data consumer with a fresh validity proof. This proof is then used by the data consumer to prove its continued access to the data set. Additionally, the TEE must have a correct view of the current wall-clock time to determine the number of times to hash the validity proof (before comparing the final result to the validity target). For the first attack, assume that adversary A ’s data-access certificate expired the previous day (for which they received a validity proof) and they now wish to gain access to that same data today by manipulating the TEE’s perception of time.

1. A submits a data-access request to the TEE containing the previous day’s validity proof X_{i-1} .
2. The TEE requests the data-access certificate from the blockchain. Assume A intercepts this message and responds with the stale (but validly signed) certificate.
3. The TEE verifies the data owner’s signature on the certificate, as well as the public identity of A contained within the certificate.
4. The TEE requests the current time from the trusted-time service.
5. Given our security assumptions, A is not able to masquerade as the time service or modify the contents of the time service’s response (without being detected); the only action A may take is to delay delivery of the time-service’s response. Thus, the TEE will always receive a time that is equal to or later than day i . Meaning, the TEE will hash X_{i-1} one (or more) times too many and the final result will not match the validity target contained in the data-access certificate.

Attack 2: Validity Proofs. For the second attack, assume that adversary A was *not* given the validity proof for the current revocation period, but still wishes to access the data set. Thus, A must gain access to the validity proof that is in the data owner’s possession or generate the proof without the help of the data owner.

1. Given our assumption that smartphones have secure storage for keys and data (e.g., validity proofs), A is not able to gain access to the data owner’s validity proofs. Furthermore, accessing the smartphone’s secure storage via bugs in the phone’s software or social engineering techniques on the phone’s user are considered out-of-scope.
2. Thus, A must generate the current revocation period’s validity proof. Given that validity proofs are generated with a cryptographically secure one-way hash function, this is not possible. Indeed, even if A has the validity proof for the previous revocation period $i - 1$, it cannot be used to figure out the proof for period i because that would require breaking the underlying hash function.

4.7.2 Identity Privacy

An adversary may try to learn private information about an individual based on their on-chain interactions (e.g., it looks like Alice is sharing data with a well-known oncologist). Under our enclave proxy model all transactions appear to originate from TEEs, however, so there is no information for the adversary to glean. The identities of data owners and consumers are stored in smart contracts – but only in encrypted form.

Attack 3: Information in Smart Contracts. Recall that blockchain users are identified with a ‘pseudonym’ (typically, a public key generated by the user) and not their real-world identity. Additionally, recall that all blockchain data (including smart-contract data) are public in the Amanuensis system.

1. Assume that adversary A learns Alice’s blockchain pseudonym. A then searches

the blockchain for all transactions that include her pseudonym. Note that while transactions are not signed by Alice (they are signed by a TEE), the transaction *data* may be a smart-contract function call where the argument(s) include her pseudonym (e.g., grant Alice access to Bob’s data).

2. *A* will not locate *any* transactions containing Alice’s pseudonym, however, as our TEE proxy encrypts pseudonyms with keys unique to each contract. Meaning, *A* would have to obtain the encryption key used on Alice’s pseudonym by stealing it from the TEE or guessing it (both of which break our security assumptions).
3. Furthermore, because there are unique keys for each contract, *A* would have to obtain keys for every contract in which it thinks Alice’s pseudonym may appear. Thus, even if *A* were able to obtain the encryption key for one contract involving Alice, it would not be able to use that information to locate other contracts involving Alice.

4.7.3 Discussion

Under the described security model, our system design ensures that attackers are incapable of (1) gaining access to sensitive mHealth data via stale data-access control certificates or (2) discerning meaningful on-chain relationships between blockchain users. Any *implementation* of our system design, however, would need to be analyzed to ensure that these properties are correctly implemented such that the privacy of blockchain users and freshness of data-access control certificates are still upheld. Indeed, there are implementation-specific attacks (e.g., side-channel attacks on Intel SGX) that could be used to break the underlying primitives on which we have built our identity privacy and data freshness solutions. For this paper, however, attacks on system implementations (e.g., side channels, software bugs, phishing) as well as those that simply aim to degrade system performance (e.g., denial-of-service) remain out-of-scope.

4.8 Evaluation

The primary function of Amanuensis is to provide confidential yet verifiable data storage and computation. To measure the impact of data confidentiality and information provenance on these operations we implemented and evaluated a prototype Amanuensis enclave. Normally, enclave applications must be coded explicitly to use the Intel SGX TEE environment, but to simplify our prototype development we used the GrapheneSGX library OS [31], which allowed us to deploy unmodified C programs as enclaves. Our implementation consists of six C programs:

- **server.c** – the Amanuensis enclave, which services data upload and computation requests from remote clients,
- **client.c** – a program that connects to the Amanuensis server enclave and simulates realistic user data upload and computation requests,
- **linreg.c** – a program that runs the linear regression algorithm on provided input data,
- **logreg.c** – a program that runs the logistic regression algorithm on provided input data,
- **ranfor.c** – a program that runs the random forest algorithm on provided input data, and
- **svm.c** – a program that runs the support vector machine algorithm on provided input data.

We evaluated our programs on a Dell OptiPlex 7060 with the following specifications:

- 6-core Intel i7-8700 CPU @ 3.2GHz
- 16GB RAM
- Ubuntu 20.03.3 LTS

- 300 Mbps wireless connection

4.8.1 Data Upload Requests

The simpler of the two operations, data upload, involves receiving data from a remote client, encrypting said data, writing it to our datastore, and generating an output provenance metadata and attestation report message. We measure the amount of CPU time taken by server.c to service data upload requests of sizes: 1KB, 10KB, 100KB, and 1MB.⁷ We ran each case twenty-five times, both for server.c running as a plain C program and for it running as an enclave, the results of which are shown in Table 4.1.

Table 4.1: The average CPU time (in milliseconds) taken by server.c to receive and save data from a remote client.

Version	1KB	10KB	100KB	1MB
plain	0.4	0.7	7.5	39.3
sgx w/o report	243.6	243.2	253.9	350.8
sgx w/report	475.3	645.7	434.5	798.9

Notice that we report average times for our SGX server both with and without the attestation report-generation phase. Such reports are generated (remotely) by the Intel Attestation Service (IAS), and the time thus depends on (1) network bandwidth and (2) the IAS response time. While most report generation times fall within 150 to 300 milliseconds, at least one iteration of every test (e.g., 1KB, 10KB) saw a report response time ranging from 1 to 5 *seconds*, which skewed the overall request time. We have no way of knowing whether such delays are a result of IAS, or the Internet, or whether such delays are common, so we include the time ‘w/o report’ to focus on aspects of performance specific to Amanuensis and SGX.

Table 4.2 depicts the average slowdown factor of data upload requests for our server when run as an enclave. I/O operations (e.g., writing files, receiving networked messages) are extremely costly for enclaves, which is why we see *less* slowdown for larger data upload

⁷We use KB and MB to refer to the numbers 2^{10} and 2^{20} , though the actual sizes of our input data sets may slightly differ from these values due to the data format requirements of our computation programs.

sizes – as we spend more time in the enclave encrypting data for storage, the overhead for reading it in begins to have less of an impact. Indeed, for upload request sizes of 1MB we only see an 8.9x slowdown (15.2x including attestation report generation) for our server when run inside an enclave versus when run as a plain C program. Larger requests amortize much of the fixed overhead. For small data upload sizes, however, the vast majority of enclave execution time is dedicated to network I/O (both for receiving the data and for receiving the attestation report).

Table 4.2: The average slowdown factor for uploading data to our server when run as an enclave versus a plain C program.

Version	1KB	10KB	100KB	1MB
sgx w/o report	627.4	329.5	34.1	8.9
sgx w/report	1137.3	603.0	58.3	15.2

4.8.2 Confidential Computation Requests

Confidential computations are a more involved process than data upload requests. We present timing measurements for the two major steps of this process (verifying the provenance of input data and executing the specified computation), as well as for the entire request itself (i.e., the combined time of the server and computation program).

4.8.3 Verifying Information Provenance

Upon receiving a computation request, the server must first verify the provenance of all input data. Specifically, for each input data set used in the computation the server must:

1. verify the IAS signature over an attestation report,
2. verify the enclave signature and code hash values,
3. verify user data hash vs. the input provenance packet,
4. fetch and decrypt the input data from the datastore,

5. verify input data set vs. the provenance packet, and
6. re-encrypt the input data and write it out to the computation's input file.

Note that the plain version of server.c differs in that it does not have to perform steps 1 and 2.

We measured the average amount of (CPU) time our server took to perform provenance verification on the following input data set configurations:

- 1000 input data sets, each with a size of 1KB
- 100 input data sets, each with a size of 10KB
- 10 input data sets, each with a size of 100KB
- 1 input data set, with a size of 1MB

Each input configuration test case was run 25 times, the results of which are presented in Table 4.3.

Table 4.3: The average CPU time (in milliseconds) and slowdown factor taken by server.c to verify the provenance of input data sets for a computation request.

Version	1KB (1000)	10KB (100)	100KB (10)	1MB (1)
plain	178.8	160.7	88.1	35.9
sgx	27,758.2	3,213.3	462.0	175.2
slowdown	155.3	20.0	5.2	4.9

The operations *directly* related to verifying the provenance of input data sets (e.g., verifying digital signatures, calculating hashes, comparing strings) incur very little overhead, as evidenced by the single 1MB data set that only sees a 4.9x slowdown between the SGX and plain version. In fact, the majority of our overhead stems from reading the data set(s) into the enclave. Even though the total amount of data for all four configurations is (relatively) the same, configurations involving many reads of small input data sets experience especially large slowdowns (155x in the worst case).

4.8.4 Executing the Computation Program

Here we present results of running real computations (linreg.c, logreg.c, ranfor.c, svm.c), each of which consists of reading and decrypting the input data file and performing a machine-learning computation. We measured the amount of CPU time taken by each program for each of the four input configurations described above. Each test case was run 25 times, the results of which are presented in Table 4.4.

Table 4.4: The average CPU time (in milliseconds) taken by the four computation programs.

Program	1KB (1000)	10KB (100)	100KB (10)	1MB (1)
linreg.c (plain)	50.5	49.4	81.7	576.2
linreg.c (sgx)	112.3	112.3	141.8	633.8
linreg.c slowdown	2.2	2.3	1.7	1.1
logreg.c (plain)	2,764.7	2,883.1	3,083.4	6,182.1
logreg.c (sgx)	6,914.7	6,986.7	7,187.3	10,445.5
logreg.c slowdown	2.5	2.4	2.3	1.7
ranfor.c (plain)	2,709.4	443.2	470.1	807.4
ranfor.c (sgx)	3,124.5	619.3	586.7	955.3
ranfor.c slowdown	1.2	1.4	1.2	1.2
svm.c (plain)	29.8	29.7	45.9	952.2
svm.c (sgx)	83.1	82.9	105.6	1,908.6
svm.c slowdown	2.8	2.8	2.3	2.0

As expected, the slowdown factors tend to trend downwards from 1KB (1000) to 1MB (1) for every computation program, but the results for the CPU time are much more varied across programs. For instance, the linreg.c program CPU time increases from 1KB (1000) to 1MB (1), whereas the ranfor.c program starts with its highest run time for the 1KB (1000) case before drastically falling and rising again at 1MB (1). The inconsistent behavior between programs for the same test cases is likely due to the fact that (1) their input data are formatted differently (e.g., one program may expect a test case with 8 integers whereas another expects 5 doubles) and (2) that they perform vastly different operations (e.g., linear regression vs. random forest). It is clear that one input data set configuration does not fit all, and that developers should profile their programs to determine which data set sizes work best. Nonetheless, the slowdowns for all of the computation programs are relatively small

compared to those seen in Section 4.8.1 and Section 4.8.3, with no program experiencing a slowdown greater than 2.8x for any input configuration.

4.8.5 Overall Performance

Thus far, we have looked at the time taken by the server to verify the provenance of input data and the time taken by the computation program itself. Now we look at the total time for the server to complete a computation request (i.e., verify input data, spawn and wait on the computation program, and generate an output provenance packet and attestation report). As we will see, the complete process includes one more large source of overhead: the time taken to instantiate the computation program.

To spawn the computation programs the server must perform a call to *fork()* and *exec()*. When running as a plain C program, this poses no issue for our server, but when run as an enclave there are serious security implications associated with these function calls. An enclave using the standard library implementations of *fork()* and *exec()* succeeds in spawning a new process to execute the computation program, but *this process executes as a plain C program*. In other words, the ‘confidential computation’ is not in fact ‘confidential’ because it is not executing in a trusted execution context.

Thankfully, the GrapheneSGX library used in our implementation wraps these system calls to ensure that the new process is instantiated as an enclave, but the cost of doing so is substantial. Indeed, we saw that on average it takes 9.1 seconds to instantiate the computation enclave. Thus, we present the overheads both for computation requests *with* and *without* the enclave instantiation time. In Table 4.5 we present the total time taken by server.c to complete a computation request for each computation under each of the four input configurations. *Note that ‘sgx1’ are times that do not include instantiation time, whereas ‘sgx2’ times do.*

Although high, the cost for instantiating an enclave is fixed. Meaning, regardless of the input configuration, it will always take ~ 9 seconds to spawn the computation program.

Table 4.5: The average CPU time (in milliseconds) taken by the server to complete computation requests for each of the computation programs.

Program	1KB (1000)	10KB (100)	100KB (10)	1MB (1)
linreg.c (plain)	229.6	210.4	169.9	612.3
linreg.c (sgx1)	28,304.5	3,971.7	1,242.5	1,251.7
linreg.c (sgx2)	37,830.1	12,926.9	10,094.7	10,184.3
logreg.c (plain)	2,941.9	3,046.2	3,169.6	6,217.8
logreg.c (sgx1)	34,809.1	10,567.8	8,091.6	11,022.5
logreg.c (sgx2)	44,306.9	19,553.2	16,897.6	19,953.4
ranfor.c (plain)	2,883.4	540.4	540.5	846.5
ranfor.c (sgx1)	31,268.8	4,282.7	1,465.7	1,535.2
ranfor.c (sgx2)	40,781.1	13,262.1	10,310.5	10,468.2
svm.c (plain)	203.5	106.1	128.8	991.8
svm.c (sgx1)	28,645.3	3,721.0	973.9	2,497.0
svm.c (sgx2)	38,170.0	12,758.5	9,857.3	11,488.5

Thus, programs with longer run times (e.g., minutes, hours) will be less impacted by this instantiation time. Indeed, the slowdown results depicted in Table 4.6 show that for our longest running program, logreg.c, our server only runs 3.2 to 15.1 times slower in an enclave than as a plain C program. If logreg.c were to run for a longer period of time – long enough to where the instantiation time becomes negligible – then we expect to see a slowdown within the range of 1.8 to 11.8 (depending on the input configuration).

Table 4.6: The average factor slowdown for the server to complete a computation request as an enclave versus as a plain C program.

Program	1KB (1000)	10KB (100)	100KB (10)	1MB (1)
linreg.c (sgx1)	123.3	18.9	7.3	2.0
linreg.c (sgx2)	164.8	61.4	59.4	16.6
logreg.c (sgx1)	11.8	3.5	2.6	1.8
logreg.c (sgx2)	15.1	6.4	5.3	3.2
ranfor.c (sgx1)	10.8	7.9	2.7	1.8
ranfor.c (sgx2)	14.1	24.5	19.1	12.4
svm.c (sgx1)	140.8	35.1	7.6	2.5
svm.c (sgx2)	187.6	120.2	76.5	11.6

4.8.6 Observations

Although even the smallest overheads incurred by SGX (e.g., 3.2x) may seem extreme to some readers, these are typical for *multi-process* programs (like `server.c`, which forks and execs other programs) executing in the SGX trusted execution context due to the high-cost of enclave instantiation and securing inter-process communications (IPC). The creators of GrapheneSGX report enclave creation times between 0.5 to 5s, but note that this time is “determined by the latency of the hardware and the Intel kernel driver, and is primarily a function of the size of the enclave” [31]. Thus, the larger the program (e.g., `linreg.c`) the longer the instantiation time; though, for a given program on a given platform, the instantiation time will remain relatively constant (as we saw with our programs over a range of input configurations).

On the other hand, the overheads for *single-process* programs (e.g., `linreg.c`) “range from matching a Linux process to less than 2x in most single-process cases” [31]. Indeed, for our single-process computations we saw slowdowns between 1.1x and 2.8x, where programs that had a higher ratio of computation time to I/O time experienced less slowdown. Based on this observation, we would expect to see smaller overall slowdowns when our `server.c` executes computations with longer run times – specifically, computations that spend more time computing than performing I/O – as it would amortize the cost of instantiation time and expensive IPC operations.

4.9 Related Work

Sensitive applications cannot take advantage of blockchain’s distributed data-management model because data and computations on the blockchain are not confidential. The need for confidentiality on the blockchain has spurred research in areas related to Zero-Knowledge Proofs (ZKPs), Multi-Party Computations (MPCs), and Trusted Execution Environments (TEEs). ZKPs and MPCs provide greater cryptographic security guarantees than TEEs, but

are computationally more expensive – making it difficult to support complex operations [9]. Amanuensis uses TEEs for data confidentiality because we want to support general computations; we thus focus on related work for TEE-enabled blockchain systems. Several works address smart-contract (i.e., computation) confidentiality by executing smart-contract functions inside trusted execution environments rather than in the mining nodes [13, 17, 54, 109]. By leveraging TEE-enabled blockchains, others use confidential smart contracts to build ML and AI data marketplaces [24, 25, 48], confidential cryptocurrency applications [8, 68, 97], smart-vehicle trust frameworks [102, 103], and secure IoT [5, 26] and healthcare [66, 102] data-management systems.

While these systems have the building blocks necessary to provide information provenance, only Liang et al. [66] explicitly attempts to do so. They take a passive approach to information provenance, however, and do not require periodic enclave attestations or verify data before it is used – data is only verified if requested, and only after the fact – whereas we perform regular attestations and verify the provenance of data before it is used. Bentov et al. [8] and Cheng et al. [17] address blockchain data freshness. Bentov et al. maintains a FIFO queue inside the enclave to store blockchain block headers, which are used to verify the correctness of blocks (and the data within) received from the blockchain. The enclave must verify *every* header in the blockchain upon startup (or from some hard-coded checkpoint), however, and communicate with multiple blockchain nodes to guarantee the correctness of the received headers. Cheng et al. flips the data freshness problem around by having the blockchain check the validity of the TEE output; if the output was generated based on stale data from the blockchain, then the blockchain miners will not add the result to the blockchain. This approach requires modifications to the consensus algorithm and blockchain software, and requires the computations to conform to their state-transition model. Many of the aforementioned systems list ‘privacy’ as one of their main goals, but often what they are actually referring to is *data* privacy (i.e., data confidentiality) and not *user* privacy (i.e., identity privacy). Only Tran et al. [97] and Kosba et al. [54] address identity privacy

as it relates to transactions on the blockchain, but both solutions are designed only for cryptocurrency applications and would not work for obscuring the types of smart-contract relationships in our system.

4.10 Conclusion

Previous work explored the use of Trusted Execution Environments (TEEs) with Blockchain technology to provide confidentiality for sensitive data and operations that would otherwise be unprotected on the open blockchain network. In this chapter, we further this work by addressing identity privacy on the blockchain, the freshness of data (access-control lists) received from the blockchain, and the provenance of confidential data for TEE-enabled blockchain systems. We expand and evaluate our Amanuensis prototype to measure the cost of information provenance and confidentiality. Although the impact of our solution is substantial at times – with slowdowns ranging from 3.2x to 188x depending on program and input configuration – applications that require confidential but verifiable computations may find these overheads acceptable (e.g., the offline processing of sensitive health data).

5

Summary and Future Directions

In this thesis we present an end-to-end solution for providing information provenance for mobile health data. In Chapter 2, we begin our investigation into information provenance by securing mHealth data at the source (i.e., on the mobile health device itself). We present a memory-isolation solution for securing sensitive mHealth applications that relies on simple Memory Protection Units (MPUs) that are available in many commodity microcontrollers. We implement and evaluate our solution on the *Amulet* [42] mHealth platform, and show that we are able to completely isolate an application’s code and memory from other programs running on the same mHealth device with a minimal impact on the device’s battery life (less than 0.5% impact).

In Chapter 3, we investigate solutions for securing mHealth data after it has left the mHealth device, which leads us to the creation of *Amanuensis*: a secure and integrated health-data system that leverages Blockchain and Trusted Execution Environment (TEE) technologies to achieve information provenance for mHealth data [38]. By using a blockchain to record and enforce data-access policies we removed the need to trust a single entity with gate-keeping the health data. Additionally, by requiring that data access and computation take place inside of TEEs, we preserve data confidentiality and provide verifiable attestations that are stored on the blockchain to support information provenance. We evaluate a prototype of Amanuensis on the VeChain Thor blockchain, and find that it is capable of supporting up to 14,256,000 mHealth data sources at \$0.47 per data source per day.

In Chapter 4, we present solutions for increasing the privacy of users transacting with the blockchain, and for verifying the authenticity and freshness of access-control lists retrieved from the blockchain. By requiring users to submit transactions through one of the many TEE servers in Amanuensis, however, we can re-write the transactions so that they appear to originate from the TEE server. Thus, user identities are hidden behind these TEEs, or “enclave proxies”, and adversaries with access to the blockchain are no longer able to discern meaningful on-chain relationships. With regards to data freshness, we combine NOVOMODO [76], a verification method for digital certificates, with a trusted third-party time service (e.g., NTS-over-TLS), to verify the authenticity and freshness of access-control lists retrieved from the blockchain. Thus, adversaries cannot feed TEEs stale access-control lists or arbitrarily delay delivery of valid lists to gain access to data outside of their allowed time period. In addition to our work on identity privacy and data freshness, we present a protocol for irrevocably tying confidential programs and their outputs (i.e., information) to Intel SGX attestations for the purpose of information provenance. We tested several real-world machine-learning applications to determine the run-time and find that each program exhibits a slowdown between 1.1-2.8x when run inside of the Intel SGX environment, and took an average of 59 milliseconds to verify the provenance of input and output data sets.

We conclude this thesis by identifying and discussing related research questions and areas that warrant further investigation.

5.1 Hardware Improvements in Ultra-Low-Power MCUs

Our memory-isolation solution employs MPU hardware to prevent applications from accessing memory outside of their boundaries, but our proof-of-concept implementation is limited by the capabilities of the MSP430 MPU (or lack thereof). Since we do not have a sufficient number of MPU memory regions, we must insert code to check for illegal memory accesses (albeit half as many checks than if we did not have the MPU at all). While there are die-space limits and complexity issues that prevent very sophisticated MPUs on constrained processors, we expect many MPUs lack functionality because microcontrollers are typically used to implement simple single-function, single-tenant embedded systems. As these ultra-low-power MCUs are increasingly being used to enable multi-application mobile devices, we recommend a greater focus be placed on developing MPUs that support a large number of memory regions (4 or more), and MPUs that are capable of protecting different kinds of memory (e.g., RAM, peripheral configuration registers). With these improvements, we would expect a drastic reduction in the run-time overhead of our solution as more MPU memory regions would remove the need for software checks and the ability to protect different memory types would allow us to reclaim use of SRAM for application stacks (as opposed to FRAM, which is slower and more energy expensive to use [49]).

5.2 Identity Privacy in Blockchain Networks

Our identity privacy solution routes transactions through enclave proxies to achieve identity privacy for users *on the blockchain*. What this solution does not address, however, is identity privacy at the *network* level. Although users have a unique pseudonym for each relationship on the blockchain, attackers can track users at the network level via their IP address to

discern on-chain relationships. Note that under our enclave proxy solution users need only contact a single enclave to submit a transaction, which in turn *broadcasts* that transaction to the blockchain network. Since users are no longer required to broadcast transactions to the blockchain network, they can take advantage of anonymous communication protocols (e.g., mix nets, DC nets, or onion routing) [93] – protocols that may be inefficient under broadcast – when communicating with an enclave.

5.3 Trusted Time Sources

Our data freshness solution verifies the freshness of access-control lists received from the blockchain network via a combination of cryptographic hash-chaining and a trusted third-party time service. We acknowledge that requiring trust in a centralized third-party time service is less than ideal, however, as it goes against the spirit of blockchain (i.e., distributing trust). There has been recent work on implementing trusted real-time clocks on (unmodified) Intel SGX hardware [64], but these solutions are still vulnerable to arbitrary delays by the OS. Indeed, we could remove the need for a trusted third-party time service by using the local hardware instead, but we would still have to fetch input data *before* requesting the time in case the OS delays delivery of the response. Thus, future work should focus on trusted real-time clock implementations that mitigate delay-of-delivery attacks. Given that the attack is made possible due to the current Intel SGX trusted architecture and computing paradigm, solutions may depend on Intel updating their hardware and microcode.

5.4 Intel SGX Side-Channels and Deprecation

Intel SGX is known to suffer from a number of side-channel attacks [92] that can be used to retrieve sensitive data (e.g., encryption keys) from enclaves. We do not address these risks with our current solution; practical implementations of Amanuensis will that use SGX will have to develop methods to limit its vulnerability to such attacks, or explore the use of other

TEE platforms. Furthermore, Intel has recently deprecated SGX beginning with its 11th and 12th generation commodity CPUs – while there was no explicit reason given, it is fair to assume that the plethora of side-channel attacks on SGX played a large part in the decision. That said, SGX will continue to be available in their server-grade CPUs (e.g., Xeon), and any businesses or organizations running hardware that pre-dates the new generation of CPUs will still have access to SGX. Therefore, we see a continued need for research related to the (safe) use of Intel SGX in data-sensitive applications and systems such as Amanuensis.

5.5 TEE Optimizations

TEEs provide confidentiality and integrity for program code and data at the cost of run-time performance, and this run-time degradation often deters those who might otherwise consider using a TEE. Thus, future work should focus on optimizing TEE run-time performance. For instance, data swapped out of RAM is first encrypted and integrity checked (likewise, data read into RAM is decrypted and integrity checked), but increasing the amount of RAM available to enclaves would reduce the number of these expensive page-swaps.

In our evaluation of the Intel SGX platform (Section 4.8), we saw that enclave instantiation time incurred significant overheads. For confidential computation programs that are run frequently, however, this overhead can be greatly reduced by keeping the enclave open even after program completion. Thus, when the program gets called again its code and data are already initialized in the TEE context; the only modifications that may need to be made are to its arguments, but this is negligible compared to instantiating a new enclave.

Typically, programs must be explicitly written to use the Intel SGX API to create, and execute in, a trusted context. To expedite our prototyping process, however, we used the GrapheneSGX library [31] which allowed us to deploy unmodified applications on the Intel SGX platform. While GrapheneSGX simplifies development, it does so at the cost of code size and run-time performance. Thus, developers may consider writing programs as

native SGX applications if run-time performance is more important than compatibility and ease-of-development.

Machine-learning and artificial-intelligence algorithms often involve highly parallelizable matrix operations and as such are often trained on GPUs, but trusted execution environments have historically only been available on CPUs. Thus, in addition to incurring the standard TEE run-time overheads, ML and AI applications cannot take advantage of GPU performance benefits if they wish to execute in a TEE context. NVIDIA recently announced their Hopper architecture for GPUs [81], however, which includes confidential-computing capabilities (e.g., confidential code and data, integrity measurement, remote attestation). Future work should focus on the combination of CPU and GPU-based TEEs to realize confidential ML and AI applications that perform on a practical time-scale.

5.6 Confidential Computation Security

Our TEE-enabled blockchain data system, Amanuensis, guarantees the correct and confidential execution of programs that compute on sensitive health data. We use ‘correct’ to refer to the integrity of the program (not the security of the overall system). Indeed, programs are executed exactly as the developers have written them; a malicious developer could write a program that simply decrypts and outputs the sensitive health data and that would be allowed under our current architecture and implementation. Thus, future work might focus on techniques for analyzing programs (e.g., language analysis, binary analysis, open-source code review) to ensure that they do not violate the security principles of our system (e.g., the confidentiality of input data) or the data-usage policies of the consortium operating Amanuensis.

5.7 Complex Data Access-Control Policies

Currently, our data access-control policies only support a simple data sharing and computation scenario. Namely, when access is granted to an mHealth data source that access is complete – all data ever produced by said source is available to the data consumer and may be used as input in any computation. Realistically, data-sharing scenarios are more nuanced and complex, however. For example, Bob may only want to grant Alice access to a subset of his data (e.g., data produced by device X during the month of May), and only for use as input in a particular computation (e.g., computation Y) [33]. Furthermore, there are legal questions that need to be answered such as “who owns the derived information?”. While this may be a legal matter, it is one that has technical implications on the design and implementation of a data-sharing system. Additionally, the syntax for expressing access-control policies given some set of rules regarding data sharing and ownership needs to be carefully considered; verifying that all combinations of access-control policies adhere to these rules may not be a trivial task.

Bibliography

- [1] S. G. Alonso, J. Arambarri, M. López-Coronado, and I. de la Torre Díez. Proposing New Blockchain Challenges in eHealth. *Journal of Medical Systems*, 43(3):64, 2019. DOI [10.1007/s10916-019-1195-7](https://doi.org/10.1007/s10916-019-1195-7). Citation on page 62.
- [2] M. P. Andersen, G. Fierro, and D. E. Culler. System Design for a Synergistic, Low Power Mote/BLE Embedded Platform. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN 2016 - Proceedings*, 2016. DOI [10.1109/IPSN.2016.7460722](https://doi.org/10.1109/IPSN.2016.7460722). Citation on page 23.
- [3] S. Angraal, H. M. Krumholz, and W. L. Schulz. Blockchain technology: Applications in health care. *Circulation: Cardiovascular Quality and Outcomes*, 10(9):1–3, 2017. DOI [10.1161/CIRCOUTCOMES.117.003800](https://doi.org/10.1161/CIRCOUTCOMES.117.003800). Citation on pages 2 and 62.
- [4] F. M. Anwar and M. Srivastava. Applications and challenges in securing time. *12th USENIX Workshop on Cyber Security Experimentation and Test, CSET 2019, co-located with USENIX Security 2019*, 2019. Citation on page 77.
- [5] G. Ayoade, V. Karande, L. Khan, and K. Hamlen. Decentralized IoT Data Management Using BlockChain and Trusted Execution Environment. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 15–22. IEEE, 7 2018. DOI [10.1109/IRI.2018.00011](https://doi.org/10.1109/IRI.2018.00011). Citation on pages 64 and 98.

- [6] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. MedRec: Using Blockchain for Medical Data Access and Permission Management. In *Proc. of the International Conference on Open and Big Data (OBD)*, pages 25–30. IEEE, 2016. DOI [10.1109/OBD.2016.11](https://doi.org/10.1109/OBD.2016.11). Citation on pages 1 and 63.
- [7] H. Bekki, T. Mackey, T. Matsuzaki, and H. Mizushima. Exploring the Potential of Blockchain Technology to Meet the Needs of 21st Century Japanese Healthcare (Preprint). *Journal of Medical Internet Research*, 22, 2019. DOI [10.2196/13649](https://doi.org/10.2196/13649). Citation on pages 2 and 62.
- [8] I. Bentov, L. Breidenbach, Y. Ji, P. Daian, F. Zhang, and A. Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1521–1538, 2019. DOI [10.1145/3319535.3363221](https://doi.org/10.1145/3319535.3363221). Citation on pages 64 and 98.
- [9] J. Bernal Bernabe, J. L. Canovas, J. L. Hernandez-Ramos, R. Torres Moreno, and A. Skarmeta. Privacy-Preserving Solutions for Blockchain: Review and Challenges. *IEEE Access*, 7:164908–164940, 2019. DOI [10.1109/ACCESS.2019.2950872](https://doi.org/10.1109/ACCESS.2019.2950872). Citation on page 98.
- [10] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms. *Mobile Networks and Applications*, 10(4):563–579, 8 2005. DOI [10.1007/s11036-005-1567-8](https://doi.org/10.1007/s11036-005-1567-8). Citation on page 22.
- [11] G. Boateng, J. A. Batsis, R. Halter, and D. Kotz. ActivityAware: An app for real-time daily activity level monitoring on the Amulet wrist-worn device. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 431–435. IEEE, 3 2017. DOI [10.1109/PERCOMW.2017.7917601](https://doi.org/10.1109/PERCOMW.2017.7917601). Citation on page 20.

- [12] G. Boateng and D. Kotz. StressAware: An app for real-time stress monitoring on the amulet wearable platform. In *2016 IEEE MIT Undergraduate Research Technology Conference (URTC)*, volume 2018-Janua, pages 1–4. IEEE, 11 2016. DOI [10.1109/URTC.2016.8284068](https://doi.org/10.1109/URTC.2016.8284068). Citation on page 20.
- [13] M. Brandenburger, C. Cachin, R. Kapitza, and A. Sorniotti. Blockchain and Trusted Computing: Problems, Pitfalls, and a Solution for Hyperledger Fabric. *arXiv*, 5 2018. Online at <http://arxiv.org/abs/1805.08541>. Citation on pages 64 and 98.
- [14] C. Brown. VeChain 11 times faster than Ethereum - Mainnet Update, 2020. Citation on page 59.
- [15] V. Buterin. Merkle in Ethereum, 2015. Online at <https://blog.ethereum.org/2015/11/15/merkle-in-ethereum/>. Citation on page 68.
- [16] Y. Chen, S. Ding, Z. Xu, H. Zheng, S. Yang, and Y. Chen. Blockchain-Based Medical Records Secure Storage and Medical Service Framework. *Journal of Medical Systems*, 43(1):5, 11 2018. DOI [10.1007/s10916-018-1121-4](https://doi.org/10.1007/s10916-018-1121-4). Citation on page 63.
- [17] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 185–200. IEEE, 6 2019. DOI [10.1109/EuroSP.2019.00023](https://doi.org/10.1109/EuroSP.2019.00023). Citation on pages 64 and 98.
- [18] X. Cheng, F. Chen, D. Xie, H. Sun, and C. Huang. Design of a Secure Medical Data Sharing Scheme Based on Blockchain. *Journal of Medical Systems*, 44(2):52, 2 2020. DOI [10.1007/s10916-019-1468-1](https://doi.org/10.1007/s10916-019-1468-1). Citation on page 63.
- [19] CoinGecko. VeThor Token (VTHO), 2020. Online at <https://www.coingecko.com/en/coins/vethor-token>. Citation on page 60.

- [20] J. Condit, M. Harren, Z. Anderson, D. Gay, and G. C. Necula. Dependent Types for Low-Level Programming. In *Lecture Notes in Computer Science*, volume 3699, pages 520–535. Springer, 2007. DOI [10.1007/978-3-540-71316-6_35](https://doi.org/10.1007/978-3-540-71316-6_35). Citation on page 22.
- [21] N. Coopridge, W. Archer, E. Eide, D. Gay, and J. Regehr. Efficient memory safety for TinyOS. In *Proceedings of the 5th international conference on Embedded networked sensor systems - SenSys '07*, (SenSys), page 205, New York, New York, USA, 2007. ACM Press. DOI [10.1145/1322263.1322283](https://doi.org/10.1145/1322263.1322283). Citation on page 22.
- [22] J. Cunningham and J. Ainsworth. Enabling patient control of personal electronic health records through distributed ledger technology. *Studies in Health Technology and Informatics*, 245:45–48, 2017. DOI [10.3233/978-1-61499-830-3-45](https://doi.org/10.3233/978-1-61499-830-3-45). Citation on page 63.
- [23] G. G. Dagher, J. Mohler, M. Milojkovic, and P. B. Marella. Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology. *Sustainable Cities and Society*, 39:283–297, 2018. DOI [10.1016/j.scs.2018.02.014](https://doi.org/10.1016/j.scs.2018.02.014). Citation on page 63.
- [24] W. Dai, C. Dai, K.-K. R. Choo, C. Cui, D. Zou, and H. Jin. SDTE: A Secure Blockchain-Based Data Trading Ecosystem. *IEEE Transactions on Information Forensics and Security*, 15:725–737, 2019. DOI [10.1109/tifs.2019.2928256](https://doi.org/10.1109/tifs.2019.2928256). Citation on pages 64 and 98.
- [25] D. Dao, D. Alistarh, C. Musat, and C. Zhang. DataBright: Towards a Global Exchange for Decentralized Data Ownership and Trusted Computation. *arXiv*, pages 1–4, 2018. Online at <http://arxiv.org/abs/1802.04780>. Citation on pages 64 and 98.

- [26] O. Dib, C. Huyart, and K. Toumi. A novel data exploitation framework based on blockchain. *Pervasive and Mobile Computing*, 61:101104, 1 2020. DOI [10.1016/j.pmcj.2019.101104](https://doi.org/10.1016/j.pmcj.2019.101104). Citation on pages 64 and 98.
- [27] M.-W. Dictionary. Amanuensis Definition. Online at <https://www.merriam-webster.com/dictionary/amanuensis>. Citation on page 4.
- [28] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462. IEEE (Comput. Soc.), 2004. DOI [10.1109/LCN.2004.38](https://doi.org/10.1109/LCN.2004.38). Citation on page 22.
- [29] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads. In *Proceedings of the 4th international conference on Embedded networked sensor systems - SenSys '06*, page 29, New York, New York, USA, 2006. ACM Press. DOI [10.1145/1182807.1182811](https://doi.org/10.1145/1182807.1182811). Citation on page 22.
- [30] C. Esposito, A. De Santis, G. Tortora, H. Chang, and K.-K. R. Choo. Blockchain: A Panacea for Healthcare Cloud-Based Data Security and Privacy? *IEEE Cloud Computing*, 5(1):31–37, 2018. DOI [10.1109/MCC.2018.011791712](https://doi.org/10.1109/MCC.2018.011791712). Citation on pages 2 and 62.
- [31] C.-C. ETsai, D. E. Porter, and M. Vij. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. *ra2017 USENIX Annual Technical Conference*, pages 645–658, 2017. Online at <https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai>. Citation on pages 58, 61, 90, 97, and 104.
- [32] K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang. MedBlock: Efficient and Secure Medical Data Sharing Via Blockchain. *Journal of Medical Systems*, 42(8):136, 2018. DOI [10.1007/s10916-018-0993-7](https://doi.org/10.1007/s10916-018-0993-7). Citation on page 63.

- [33] E. Greene, P. Proctor, and D. Kotz. Secure sharing of mHealth data streams through cryptographically-enforced access control. *Smart Health*, 2018. DOI [10.1016/j.smhl.2018.01.003](https://doi.org/10.1016/j.smhl.2018.01.003). Citation on pages 62 and 106.
- [34] N. W. Group. Network Time Security for the Network Time Protocol. Online at <https://tools.ietf.org/id/draft-ietf-ntp-using-nts-for-ntp-10.html>. Citation on page 77.
- [35] L. Gu and J. A. Stankovic. t-kernel. In *Proceedings of the 4th international conference on Embedded networked sensor systems - SenSys '06*, SenSys '06, page 1, New York, New York, USA, 2006. ACM Press. DOI [10.1145/1182807.1182809](https://doi.org/10.1145/1182807.1182809). Citation on page 22.
- [36] C.-c. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. A dynamic operating system for sensor nodes. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services - MobiSys '05*, page 163, New York, New York, USA, 2005. ACM Press. DOI [10.1145/1067170.1067188](https://doi.org/10.1145/1067170.1067188). Citation on page 22.
- [37] T. Hardin and D. Kotz. Blockchain in Health Data Systems: A Survey. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 490–497. IEEE, 10 2019. DOI [10.1109/IOTSMS48152.2019.8939174](https://doi.org/10.1109/IOTSMS48152.2019.8939174). Citation on page 62.
- [38] T. Hardin and D. Kotz. Amanuensis: Information provenance for health-data systems. *Information Processing & Management*, 58(2):102460, 3 2021. DOI [10.1016/j.ipm.2020.102460](https://doi.org/10.1016/j.ipm.2020.102460). Citation on pages 4, 67, 69, and 101.
- [39] A. Hattendorf, A. Raabe, and A. Knoll. Shared memory protection for spatial separation in multicore architectures. In *7th IEEE International Symposium on Industrial Embedded Systems, SIES 2012 - Conference Proceedings*, pages 299–302, 2012. DOI [10.1109/SIES.2012.6356601](https://doi.org/10.1109/SIES.2012.6356601). Citation on page 22.

- [40] R. Henry, A. Herzberg, and A. Kate. Blockchain access privacy: Challenges and directions. *IEEE Security and Privacy*, 16(4):38–45, 2018. DOI [10.1109/MSP.2018.3111245](https://doi.org/10.1109/MSP.2018.3111245). Citation on page 26.
- [41] J. Herrera-Joancomartí. Research and Challenges on Bitcoin Anonymity. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, pages 3–16, 2015. DOI [10.1007/978-3-319-17016-9_{-}1](https://doi.org/10.1007/978-3-319-17016-9_{-}1). Citation on page 68.
- [42] J. Hester, T. Peters, T. Yun, R. Peterson, J. Skinner, B. Golla, K. Storer, S. Hearndon, K. Freeman, S. Lord, R. Halter, D. Kotz, and J. Sorber. Amulet. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, pages 216–229, New York, NY, USA, 11 2016. ACM. DOI [10.1145/2994551.2994554](https://doi.org/10.1145/2994551.2994554). Citation on pages 3, 8, 9, 10, 22, and 100.
- [43] N. Ho and A. V. Dinh-Duc. MemMON: Run-time off-chip detection for memory access violation in embedded systems. In *ACM International Conference Proceeding Series*, pages 114–121, 2010. DOI [10.1145/1852611.1852634](https://doi.org/10.1145/1852611.1852634). Citation on page 22.
- [44] M. Hölbl, M. Kompara, A. Kamišalić, and L. N. Zlatolas. A systematic review of the use of blockchain in healthcare. *Symmetry*, 10(10), 2018. DOI [10.3390/sym10100470](https://doi.org/10.3390/sym10100470). Citation on page 62.
- [45] P. Hooda. Blockchain Forks. Online at <https://www.geeksforgeeks.org/blockchain-forks/>. Citation on page 74.
- [46] H. M. Hussien, S. M. Yasin, S. N. I. Udzir, A. A. Zaidan, and B. B. Zaidan. A Systematic Review for Enabling of Develop a Blockchain Technology in Healthcare Application: Taxonomy, Substantially Analysis, Motivations, Challenges, Recommendations and Future Direction. *Journal of Medical Systems*, 43(10):320, 10 2019. DOI [10.1007/s10916-019-1445-8](https://doi.org/10.1007/s10916-019-1445-8). Citation on page 62.

- [47] R. H. Hylock and X. Zeng. A Blockchain Framework for Patient-Centered Health Records and Exchange (HealthChain): Evaluation and Proof-of-Concept Study. *Journal of Medical Internet Research*, 21(8):e13592, 8 2019. DOI [10.2196/13592](https://doi.org/10.2196/13592). Citation on page 63.
- [48] N. Hynes, D. Dao, D. Yan, R. Cheng, and D. Song. A demonstration of sterling: A privacy-preserving data marketplace. *Proceedings of the VLDB Endowment*, 11(12):2086–2089, 2018. DOI [10.14778/3229863.3236266](https://doi.org/10.14778/3229863.3236266). Citation on pages 64 and 98.
- [49] T. INSTRUMENTS. MSP430FR5969 16 mhz ultra-low-power microcontroller featuring 64 kb fram, 2kb sram, 40 io,, 2017. Citation on pages 11 and 102.
- [50] Intel. Intel Software Guard Extensions. Online at <https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html>. Citation on pages 57 and 66.
- [51] A. Khan, A. Schäfer, and M. Zetlmeisl. Efficient memory-protected integration of add-on software subsystems in small embedded automotive applications. *IEEE Transactions on Industrial Informatics*, 3(1), 2007. DOI [10.1109/TII.2006.890522](https://doi.org/10.1109/TII.2006.890522). Citation on page 23.
- [52] A. Khatoon. A Blockchain-Based Smart Contract System for Healthcare Management. *Electronics*, 9(1):94, 1 2020. DOI [10.3390/electronics9010094](https://doi.org/10.3390/electronics9010094). Citation on page 63.
- [53] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan. TrustLite. In *Proceedings of the Ninth European Conference on Computer Systems - EuroSys '14*, EuroSys '14, pages 1–14, New York, New York, USA, 4 2014. ACM Press. DOI [10.1145/2592798.2592824](https://doi.org/10.1145/2592798.2592824). Citation on page 22.
- [54] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *Proceedings -*

- 2016 *IEEE Symposium on Security and Privacy, SP 2016*, pages 839–858. Institute of Electrical and Electronics Engineers Inc., 8 2016. DOI [10.1109/SP.2016.55](https://doi.org/10.1109/SP.2016.55). Citation on pages 64 and 98.
- [55] J. P. Ku and I. Sim. Mobile Health: making the leap to research and clinics. *npj Digital Medicine*, 4(1):83, 12 2021. DOI [10.1038/s41746-021-00454-z](https://doi.org/10.1038/s41746-021-00454-z). Citation on page 1.
- [56] R. Kumar, E. Kohler, and M. Srivastava. Harbor. In *Proceedings of the 6th international conference on Information processing in sensor networks - IPSN '07*, page 340, New York, New York, USA, 2007. ACM Press. DOI [10.1145/1236360.1236404](https://doi.org/10.1145/1236360.1236404). Citation on page 22.
- [57] P. Levis and D. Culler. Maté. *ACM SIGPLAN Notices*, 37(10):85–95, 10 2002. DOI [10.1145/605432.605407](https://doi.org/10.1145/605432.605407). Citation on page 22.
- [58] P. Levis, D. Gay, and D. Culler. Active Sensor Networks. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2, NSDI'05*, pages 343–356, Berkeley, CA, USA, 2005. USENIX Association. Online at <http://dl.acm.org/citation.cfm?id=1251203.1251228>. Citation on page 22.
- [59] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. TinyOS: An Operating System for Sensor Networks. In *Ambient Intelligence*, volume 35, pages 115–148. Springer-Verlag, Berlin/Heidelberg, 2005. DOI [10.1007/3-540-27139-2_{_}7](https://doi.org/10.1007/3-540-27139-2_{_}7). Citation on page 22.
- [60] A. Levy, B. Campbell, B. Ghena, D. B. Giffin, P. Pannuto, P. Dutta, and P. Levis. Multiprogramming a 64kB Computer Safely and Efficiently. In *Proceedings of the 26th Symposium on Operating Systems Principles, SOSP '17*, pages 234–251, New York, NY, USA, 10 2017. ACM. DOI [10.1145/3132747.3132786](https://doi.org/10.1145/3132747.3132786). Citation on page 22.

- [61] G. Li and S. Top. Towards spatial isolation design in a multi-core real-time kernel targeting safety-critical applications. *International Journal of Critical Computer-Based Systems*, 4(3), 2013. DOI [10.1504/IJCCBS.2013.058402](https://doi.org/10.1504/IJCCBS.2013.058402). Citation on page 22.
- [62] H. Li, L. Zhu, M. Shen, F. Gao, X. Tao, and S. Liu. Blockchain-Based Data Preservation System for Medical Data. *Journal of Medical Systems*, 42(8):141, 2018. DOI [10.1007/s10916-018-0997-3](https://doi.org/10.1007/s10916-018-0997-3). Citation on page 63.
- [63] J. Li, J. Wu, G. Jiang, and T. Srikanthan. Blockchain-based public auditing for big data in cloud storage. *Information Processing and Management*, 57(6):102382, 2020. DOI [10.1016/j.ipm.2020.102382](https://doi.org/10.1016/j.ipm.2020.102382). Citation on page 63.
- [64] H. Liang, M. Li, Y. Chen, T. Yang, Z. Xie, and L. Jiang. Architectural Protection of Trusted System Services for SGX Enclaves in Cloud Computing. *IEEE Transactions on Cloud Computing*, 9(3):910–922, 2021. DOI [10.1109/TCC.2019.2892449](https://doi.org/10.1109/TCC.2019.2892449). Citation on page 103.
- [65] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla. ProvChain: A Blockchain-Based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 468–477. IEEE, 5 2017. DOI [10.1109/CCGRID.2017.8](https://doi.org/10.1109/CCGRID.2017.8). Citation on page 63.
- [66] X. Liang, S. Shetty, J. Zhao, D. Bowden, D. Li, and J. Liu. Towards decentralized accountability and self-sovereignty in healthcare systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018. DOI [10.1007/978-3-319-89500-0_{_}34](https://doi.org/10.1007/978-3-319-89500-0_{_}34). Citation on pages 1, 64, and 98.

- [67] X. Liang, J. Zhao, S. Shetty, J. Liu, and D. Li. Integrating blockchain for data sharing and collaboration in mobile healthcare applications. In *Proc. of the International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–5. IEEE, 2017. DOI [10.1109/PIMRC.2017.8292361](https://doi.org/10.1109/PIMRC.2017.8292361). Citation on page 63.
- [68] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch. Teechain. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles - SOSP '19*, pages 63–79, New York, New York, USA, 2019. ACM Press. DOI [10.1145/3341301.3359627](https://doi.org/10.1145/3341301.3359627). Citation on pages 64 and 98.
- [69] J. Liu, X. Li, L. Ye, H. Zhang, X. Du, and M. Guizani. BPDS: A Blockchain Based Privacy-Preserving Data Sharing for Electronic Medical Records. In *2018 IEEE Global Communications Conference, GLOBECOM 2018 - Proceedings*, pages 1–6. Institute of Electrical and Electronics Engineers Inc., 12 2018. DOI [10.1109/GLOCOM.2018.8647713](https://doi.org/10.1109/GLOCOM.2018.8647713). Citation on page 63.
- [70] Y.-S. Lo, C.-Y. Yang, H.-F. Chien, S.-S. Chang, C.-Y. Lu, and R.-J. Chen. Blockchain-enabled iWellChain Framework Integration With the National Medical Referral System: Implementation and Preliminary Results (Preprint). *Journal of Medical Internet Research*, 21(12):1–13, 12 2019. DOI [10.2196/13563](https://doi.org/10.2196/13563). Citation on page 63.
- [71] L. Lopriore. Hardware support for memory protection in sensor nodes. *Microprocessors and Microsystems*, 38(3):226–232, 2014. DOI [10.1016/j.micpro.2014.01.004](https://doi.org/10.1016/j.micpro.2014.01.004). Citation on page 22.
- [72] L. Lopriore. Memory protection in embedded systems. *Journal of Systems Architecture*, 63, 2016. DOI [10.1016/j.sysarc.2016.01.006](https://doi.org/10.1016/j.sysarc.2016.01.006). Citation on page 22.
- [73] S. Lu. VeChain THOR Power Forged AMA, 2018. Online at <https://youtu.be/IWoEsBQFozM?t=438>. Citation on page 59.

- [74] A. Mense and L. Athanasiadis. Concept for Sharing Distributed Personal Health Records with Blockchains. *Studies in Health Technology and Informatics*, 251:7–10, 2018. DOI [10.3233/978-1-61499-880-8-7](https://doi.org/10.3233/978-1-61499-880-8-7). Citation on page 63.
- [75] L. Mertz. (Block) Chain Reaction: A Blockchain Revolution Sweeps into Health Care, Offering the Possibility for a Much-Needed Data Solution. *IEEE Pulse*, 2018. DOI [10.1109/MPUL.2018.2814879](https://doi.org/10.1109/MPUL.2018.2814879). Citation on pages 2 and 62.
- [76] S. Micali. NOVOMODO: Scalable certificate validation and simplified pki management. *1st Annual PKI Research Workshop*, page 15, 2002. Citation on pages 5 and 101.
- [77] T. Mikula and R. H. Jacobsen. Identity and access management with blockchain in electronic healthcare records. In *Proceedings - 21st Euromicro Conference on Digital System Design, DSD 2018*, pages 699–706. Institute of Electrical and Electronics Engineers Inc., 10 2018. DOI [10.1109/DSD.2018.00008](https://doi.org/10.1109/DSD.2018.00008). Citation on page 63.
- [78] F. Mohammadi, A. Panou, C. Ntantogian, E. Karapistoli, E. Panaousis, and C. Xenakis. CUREX: Secure and private health data exchange. *Proceedings - 2019 IEEE/WIC/ACM International Conference on Web Intelligence Workshops, WI 2019 Companion*, pages 263–268, 2019. DOI [10.1145/3358695.3361753](https://doi.org/10.1145/3358695.3361753). Citation on page 63.
- [79] M. Möser and R. Böhme. The price of anonymity: Empirical evidence from a market for Bitcoin anonymization. *Journal of Cybersecurity*, 3(2):127–135, 2017. DOI [10.1093/cybsec/tyx007](https://doi.org/10.1093/cybsec/tyx007). Citation on page 68.
- [80] T. Nugent, D. Upton, and M. Cimpoesu. Improving data transparency in clinical trials using blockchain smart contracts. *F1000Research*, 5:2541, 10 2016. DOI [10.12688/f1000research.9756.1](https://doi.org/10.12688/f1000research.9756.1). Citation on page 63.

- [81] NVIDIA. NVIDIA Announces Hopper Architecture, the Next Generation of Accelerated Computing, 2022. Online at <https://nvidianews.nvidia.com/news/nvidia-announces-hopper-architecture-the-next-generation-of-accelerated-computing>. Citation on page 105.
- [82] OrthoLive. 32 Statistics on mHealth. Online at <https://www.ortholive.com/blog/32-statistics-on-mhealth/>. Citation on page 1.
- [83] J. Porquet, A. Greiner, and C. Schwarz. NoC-MPU: A secure architecture for flexible co-hosting on shared memory MPSoCs. In *Proceedings -Design, Automation and Test in Europe, DATE*, pages 591–594, 2011. DOI [10.1109/date.2011.5763291](https://doi.org/10.1109/date.2011.5763291). Citation on page 22.
- [84] I. Radanović and R. Likić. Opportunities for Use of Blockchain Technology in Medicine. *Applied Health Economics and Health Policy*, 16(5):583–590, 2018. DOI [10.1007/s40258-018-0412-8](https://doi.org/10.1007/s40258-018-0412-8). Citation on pages 2 and 62.
- [85] F. Reid and M. Harrigan. An Analysis of Anonymity in the Bitcoin System. In Y. Altshuler, Y. Elovici, A. B. Cremers, N. Aharony, and A. Pentland, editors, *Security and Privacy in Social Networks*, pages 197–223. Springer New York, New York, NY, 2013. DOI [10.1007/978-1-4614-4139-7_{_}10](https://doi.org/10.1007/978-1-4614-4139-7_{_}10). Citation on page 68.
- [86] Research and Markets. Global Mobile Health App Market 2021 to 2026: Increasing Smartphone and Internet Penetration to Augment the Market Size, 2021. Online at <https://www.globenewswire.com/news-release/2021/12/23/2357205/28124/en/Global-Mobile-Health-App-Market-2021-to-2026-Increasing-Smartphone-and-Internet-Penetration.html>. Citation on page 1.
- [87] J. G. Rivera. Hardware-based data protection/isolation at runtime in Ada code for microcontrollers. In *Ada User Journal*, volume 38, 2017. DOI [10.1145/3232693.3232705](https://doi.org/10.1145/3232693.3232705). Citation on page 23.

- [88] J. M. Roman-Belmonte, H. De la Corte-Rodriguez, and E. C. Rodriguez-Merchan. How blockchain technology can change medicine. *Postgraduate Medicine*, 130(4):420–427, 2018. DOI [10.1080/00325481.2018.1472996](https://doi.org/10.1080/00325481.2018.1472996). Citation on pages 2 and 62.
- [89] P. Ruan, G. Chen, T. T. A. Dinh, Q. Lin, B. C. Ooi, and M. Zhang. Fine-grained, secure and efficient data provenance on blockchain systems. *Proceedings of the VLDB Endowment*, 12(9):975–988, 5 2019. DOI [10.14778/3329772.3329775](https://doi.org/10.14778/3329772.3329775). Citation on page 63.
- [90] M. Sabt, M. Achemlal, and A. Bouabdallah. Trusted Execution Environment: What It is, and What It is Not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64. IEEE, 8 2015. DOI [10.1109/Trustcom.2015.357](https://doi.org/10.1109/Trustcom.2015.357). Citation on page 27.
- [91] F. Sant’Anna, N. Rodriguez, R. Ierusalimsky, O. Landsiedel, and P. Tsigas. Safe system-level concurrency on resource-constrained nodes. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems - SenSys ’13*, pages 1–14, New York, New York, USA, 2013. ACM Press. DOI [10.1145/2517351.2517360](https://doi.org/10.1145/2517351.2517360). Citation on page 22.
- [92] M. Schwarz and D. Gruss. How Trusted Execution Environments Fuel Research on Microarchitectural Attacks. *IEEE Security & Privacy*, 18(5):18–27, 9 2020. DOI [10.1109/MSEC.2020.2993896](https://doi.org/10.1109/MSEC.2020.2993896). Citation on page 103.
- [93] F. Shirazi, M. Simeonovski, M. R. Asghar, M. Backes, and C. Diaz. A Survey on Routing in Anonymous Communication Protocols. *ACM Computing Surveys*, 51(3):1–39, 5 2019. DOI [10.1145/3182658](https://doi.org/10.1145/3182658). Citation on page 103.
- [94] J. Sriram, M. Shin, D. Kotz, A. Rajan, M. Sastry, and M. Yarvis. Challenges in Data Quality Assurance in Pervasive Health Monitoring Systems. *Future of Trust in Computing: Proceedings of the First International Conference Future of Trust*

- in Computing 2008: With 58 Illustrations*, pages 129–142, 2009. DOI [10.1007/978-3-8348-9324-6{_}14](#). Citation on page 3.
- [95] O. Stecklina, P. Langendoerfer, and H. Menzel. Design of a tailor-made memory protection unit for low power microcontrollers. In *Proceedings of the 8th IEEE International Symposium on Industrial Embedded Systems, SIES 2013*, pages 225–231, 2013. DOI [10.1109/SIES.2013.6601495](#). Citation on page 22.
- [96] S. R. Steinhubl, E. D. Muse, and E. J. Topol. The emerging field of mobile health. *Science Translational Medicine*, 7(283):1–12, 2015. DOI [10.1126/scitranslmed.aaa3487](#). Citation on page 1.
- [97] M. Tran, L. Luu, M. Suk Kang, I. Bentov, and P. Saxena. Obscuro: A Bitcoin Mixer using Trusted Execution Environments. *ACSAC '18 (Annual Computer Security Applications Conference)*, 18:692–701, 2018. DOI [10.1145/3274694.3274750](#). Citation on pages 64 and 98.
- [98] M. Uddin, A. Stranieri, I. Gondal, and V. Balasubramanian. Blockchain Leveraged Decentralized IoT eHealth Framework. *Internet of Things*, 9:100159, 2020. DOI [10.1016/j.iot.2020.100159](#). Citation on page 63.
- [99] M. A. Uddin, A. Stranieri, I. Gondal, and V. Balasubramanian. Continuous Patient Monitoring With a Patient Centric Agent: A Block Architecture. *IEEE Access*, 6:32700–32726, 2018. DOI [10.1109/ACCESS.2018.2846779](#). Citation on page 63.
- [100] A. A. Vazirani, O. O’Donoghue, D. Brindley, and E. Meinert. Implementing Blockchains for Efficient Health Care: Systematic Review. *Journal of Medical Internet Research*, 21(2):e12439, 2019. DOI [10.2196/12439](#). Citation on page 62.
- [101] VeChain Foundation. VeChain Whitepaper 2.0. Technical report, VeChain Foundation, 2019. Online at <https://www.vechain.org/whitepaper>. Citation on page 57.

- [102] C. M. Wachira, S. L. Remy, O. Bent, N. Bore, S. Osebe, K. Weldemariam, and A. Walcott-Bryant. A Platform for Disease Intervention Planning. In *2020 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 1–8. IEEE, 11 2020. DOI [10.1109/ICHI48887.2020.9374384](https://doi.org/10.1109/ICHI48887.2020.9374384). Citation on page 98.
- [103] D. Wang, X. Chen, H. Wu, R. Yu, and Y. Zhao. A blockchain-based vehicle-trust management framework under a crowdsourcing environment. *Proceedings - 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2020*, pages 1950–1955, 2020. DOI [10.1109/TrustCom50675.2020.00266](https://doi.org/10.1109/TrustCom50675.2020.00266). Citation on page 98.
- [104] H. Wang and Y. Song. Secure Cloud-Based EHR System Using Attribute-Based Cryptosystem and Blockchain. *Journal of Medical Systems*, 42(8):152, 2018. DOI [10.1007/s10916-018-0994-6](https://doi.org/10.1007/s10916-018-0994-6). Citation on page 63.
- [105] G. Wood. Ethereum: a secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, pages 1–32, 2014. DOI [10.1017/CBO9781107415324.004](https://doi.org/10.1017/CBO9781107415324.004). Citation on page 57.
- [106] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani. MeDShare: Trust-Less Medical Data Sharing Among Cloud Service Providers via Blockchain. *IEEE Access*, 5:14757–14767, 7 2017. DOI [10.1109/ACCESS.2017.2730843](https://doi.org/10.1109/ACCESS.2017.2730843). Citation on page 63.
- [107] J. Xu, K. Xue, S. Li, H. Tian, J. Hong, P. Hong, and N. Yu. Healthchain: A Blockchain-based Privacy Preserving Scheme for Large-scale Health Data. *IEEE Internet of Things Journal*, 6(5):1–1, 2019. DOI [10.1109/jiot.2019.2923525](https://doi.org/10.1109/jiot.2019.2923525). Citation on page 63.
- [108] J. Yiu and M. Meriac. High-End Security Features for Low-End Microcontrollers. In *Proceedings of Embedded World*, 2017. Citation on page 23.

- [109] R. Yuan, Y.-B. Xia, H.-B. Chen, B.-Y. Zang, and J. Xie. ShadowEth: Private Smart Contract on Public Blockchain. *Journal of Computer Science and Technology*, 33(3):542–556, 5 2018. DOI [10.1007/s11390-018-1839-y](https://doi.org/10.1007/s11390-018-1839-y). Citation on pages 64 and 98.
- [110] Q. Zhao, S. Chen, Z. Liu, T. Baker, and Y. Zhang. Blockchain-based privacy-preserving remote data integrity checking scheme for IoT information systems. *Information Processing and Management*, 57(6):102355, 2020. DOI [10.1016/j.ipm.2020.102355](https://doi.org/10.1016/j.ipm.2020.102355). Citation on page 63.
- [111] A. Zuepke, K. Beckmann, A. Zoor, and R. Kroeger. For Safety and Security Reasons: The Cost of Component-Isolation in IoT. *Logistics & Sustainable Transport*, 7(1):41–50, 10 2016. DOI [10.1515/jlst-2016-0004](https://doi.org/10.1515/jlst-2016-0004). Citation on page 23.