SocIoTy: Practical Cryptography in Smart Home Contexts

Tushar M. Jois
City College of New York
tjois@ccny.cuny.edu

Gabrielle Beck
Johns Hopkins University
becgabri@cs.jhu.edu

Sofia Belikovetsky
Johns Hopkins University
sofia.belikovetsky@gmail.com

Joseph Carrigan
Johns Hopkins University
joseph.carrigan@jhu.edu

Alishah Chator
Boston University
alishahc@bu.edu

Logan Kostick
Johns Hopkins University
lkostic1@jhu.edu

Maximilian Zinkus
Johns Hopkins University
zinkus@cs.jhu.edu

Gabriel Kaptchuk
Boston University
kaptchuk@bu.edu

Aviel D. Rubin
Johns Hopkins University
rubin@cs.jhu.edu

ABSTRACT

Smartphones form an important source of trust in modern computing. But, while their mobility is convenient, smartphones can be stolen or seized, allowing an adversary to impersonate the user in their digital life: accessing the user's services and decrypting their sensitive files. With this in mind, we build *SocIoTy*, which leverages a user's existing IoT devices to add a context-sensitive layer of security for non-expert users. Instead of assuming the existence of dedicated hardware, SocIoTy re-uses the devices of a user's smart home to provide cryptographic services, which we term *at-home cryptography*. We show that at-home cryptography can be built from simple cryptographic primitives, and that our SocIoTy solution is able to provide useful functionalities, like two-factor authentication (2FA) and secure file storage, while protecting against powerful adversaries in this setting. We implement and evaluate SocIoTy in real-world use cases and provide microbenchmarks for individual cryptographic operations on realistic models of IoT devices. We also provide full benchmarks of an end-to-end deployment on a simulated smart home, using a smartphone and 9 IoT devices to generate and display 2FA one-time passwords in less than 200 milliseconds. SocIoTy is able to provide strong, practical cryptography while binding its execution to the smart home itself, all without requiring additional hardware.

KEYWORDS

smart home computation, context-based cryptography, two-factor authentication, systems security, compelled access security

1 INTRODUCTION

Mobile devices have quickly become users' most important trusted computing base. Users rely on them to authenticate and interact with services that perform sensitive tasks, *e.g.*, online banking, file storage, and telehealth. These tasks are often secured using a combination of passwords and locally-stored cryptographic secrets, *e.g.*, one-time passwords (OTPs) generated by a smartphone app.

The convenience afforded by mobile computing is accompanied by a commensurate increase in risk. Mobile devices are highly portable, allowing them to be easily lost or stolen. Once a mobile device is taken, any cryptographic material on the device could be extracted [73]. This would allow an adversary to impersonate the user and access their services—a catastrophic breakdown in online security and privacy. A corporate spy, for instance, could use extracted 2FA OTPs to connect to a rival company's internal VPN. This threat is particularly dire when the user needs to keep their data private from law enforcement agencies with access to software that can be used to circumvent on-device security measures. For example, border police could decrypt files from a user's cloud storage, inspecting it for content deemed subversive.

Mitigating risk with at-home cryptography. To mitigate the risk posed by device loss, users should be able to *voluntarily* restrict access to critical key material to times when their device is in some trustworthy *context*, *e.g.*, when at home. These users can *opt-in* to restricting their usage, possibly because they may consider a certain subset of actions too sensitive to operate outside of a secure context, or might consider themselves particularly at-risk. For instance, users might browse social media wherever they are, but may already limit their use of online banking or telemedicine to times when they are at home for privacy reasons. By limiting their use in this way, these users actively engage in misuse resistance for their critical services, protecting themselves from risks outside of the home.

Users may also wish to utilize recently proposed privacy enhancing systems that assume the existence of a personal, fixed storage for secrets local to a user. For example, BurnBox [65] provides *self-revocable* encryption, which allows users to temporarily delete keys that could decrypt sensitive cloud data (*e.g.*, before a border crossing), and recovers these keys after the user is safely home with key material stored there.

The fundamental building block required to realize these applications is computation that can only be performed at home, which can then be leveraged to perform cryptographic operations. We refer to such a system as one that provides *at-home cryptography*.

Importantly, mobile devices cannot facilitate at-home cryptography alone, as they cannot offer fine-grained context-sensitive access control mechanisms. Even if the user only uses their material in the home context, this material is still on device—and therefore

Proceedings on Privacy Enhancing Technologies 2024(1), 447–464

https://doi.org/10.56553/popets-2024-0026

exposed—while they are on the go. Similarly, mobile devices cannot be the fixed store of secrets required for advanced privacy systems. Thus, mobile devices need assistance from other parts of the physical environment to enforce context sensitivity.

Prior attempts. At first glance, achieving limited access to keys might appear trivial; a user can simply store key material for authentication or encryption on a device that *stays* at home rather than on their mobile device, *e.g.*, a desktop computer or a dedicated hardware security module. Because this device is stationary and only accessible on a local network—or even air-gapped—access to the key material is inherently limited.

While straightforward, this solution requires the user to (1) own stationary hardware, and (2) have the technical expertise to manage their stationary hardware. In the time before widespread smartphone use, this solution made sense; personal computers were not very portable, and required at least some level of technical expertise to operate. However, this ostensibly simple solution is becoming unworkable for a rising part of the general population. 15% of adults in the United States only use smartphones as their primary device, with an upward trend since 2013; in the youngest generation of adults, 18-29 years old, this proportion increases to 28% [51].

Past research efforts also focus on the use of dedicated hardware to build this functionality. While *context-sensitive cryptography* (of which at-home cryptography is a subset) has been studied in both the theoretical [13, 15, 17, 34, 50] and applied [6, 43, 60–62, 66] literature, no practical constructions have been realized or widely deployed.[1] As such, we turn to a more pragmatic approach: *re-use* the devices users already have to establish a context.

Re-using devices for at-home cryptography. Although users are unlikely to have access to dedicated hardware for at-home cryptography, users may have access to Internet-of-Things (IoT) devices. These devices typically do not leave the home, making them an attractive prospect for anchoring a trusted computing base for at-home cryptography.

Re-using these devices raises its own difficulties, however:

- IoT devices are designed to be *single-purpose*, and, to keep costs low, have just enough compute capability to provide their application, unlike the general purpose capabilities of smartphones and computers. Re-using an IoT device beyond its intended purpose may induce measurable overhead, so we must ensure that at-home cryptography operations are lightweight.
- IoT devices have a *history of vulnerabilities* [23, 58], so it is not advisable to use one as a single store of secrets. Instead, we observe that it is more appropriate to distribute trust among many IoT devices, such that an attacker would need to compromise many IoT devices before exposing any of the user's secret data. Of course, resource constraints limit the viable approaches to federating trust.

As such, leveraging IoT devices into a practical at-home cryptography system requires carefully navigating tradeoffs between functionality, deployability, and security.

SocIoTy. In this work we present *SocIoTy*, a system design and protocol for at-home cryptography using a user's existing IoT devices. SocIoTy is designed for non-expert users who want to protect high-value digital resources from powerful, privacy-invading adversaries but do not have the access, expertise, or inclination to use dedicated hardware. Our system allows these users to set their own risk tolerances, allowing them to tie whatever secrets they consider to be most valuable to their smart home. Unlike existing privacy-enhancing systems, SocIoTy protects against common surveillance techniques even if the user's devices are compromised (like during border searches). While we focus on the smart home in this work, SocIoTy has applications wherever there are multiple embedded devices running on the same network, such as small businesses and hospitals. Our design is summarized in Figure 1.

SocIoTy builds an at-home cryptographic system for a pseudorandom function (PRF) [27], a simple, but powerful, primitive. From this at-home PRF, we can directly build two-factor authentication [45, 46] and derive keys for encryption. SocIoTy treats the smart home as a PRF that users can query to provide at-home cryptographic services. Because the user is physically at home, they can generate PRF outputs, and use these outputs to address their real-world needs—like generation of 2FA OTPs for authentication and of keys for cloud-encrypted content—all without worrying that their credentials are at risk outside of the home. Moreover, the interface to users is the same, and service providers would not have to change their architectures to accommodate SocIoTy; users perform one setup step on their smart home, and service providers only need to use a different PRF library in their backends.

To address the problems of IoT devices discussed above, we build a "dual-layered" PRF where one layer is produced by the smart home, and the other by the a more powerful device (*e.g.*, a smartphone). This dual-layered PRF is realized by combining the output of a threshold, distributed PRF [47] (TDPRF) with that of a normal PRF. Each IoT device computes a partial evaluation on an input, and the more powerful device reconstructs the TDPRF result from the smart home's multiple partial evaluations and combines it with a PRF evaluation computed on its own key material. This federates trust among all of the participants in the smart home: the individual IoT devices along with the smartphone that controls them. Any compromising party would be forced to corrupt both layers to recover the final output. At the same time, the computation power required by the IoT devices is low, since only one operation is needed (in implementation, a single elliptic curve multiplication) per each user request.

We evaluate SocIoTy on a simulated smart home, consisting of analogs of smart home devices, from high-end, full-size systems (Raspberry Pis) to tiny, embedded microcontrollers (ESP32s). We collect microbenchmarks on these devices, as well as benchmarks on full deployments of the system in realistic configurations. To highlight the ease of use of our proposed system, we also build a simple Google Authenticator-style smartphone app that uses SocIoTy to calculate OTPs. We find that our implementation meets the performance needs of our envisioned applications, while remaining seamless to the end user—performing OTP generation, for example, in <200 milliseconds on average when involving a smartphone and 9 SocIoTy devices.

Contributions. In this work, we study the problem of giving non-expert users context-sensitive access control to their cryptographic material, focusing on the smart home setting. Our goal is to help

[1] Indeed, there are impossibility results that might rule out "ideal" solutions [17].

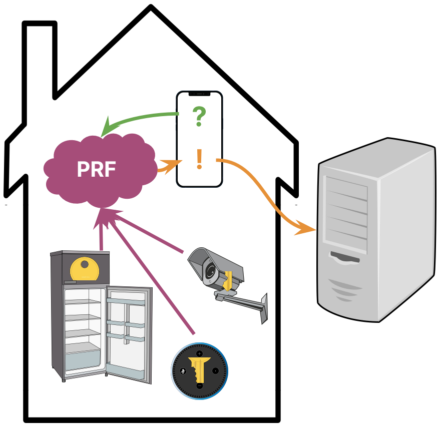


Figure 1: An overview of SocIoTy, which uses a PRF built from IoT devices to provide at-home cryptographic services.

average users mitigate the risk associated with carrying high-value cryptographic material on their mobile devices, giving users the peace-of-mind in knowing that their service or files can only be accessed from home. Specifically,

(1) We discuss *at-home cryptography*, highlighting relevant use cases and design considerations for a context-sensitive cryptographic system at home (Section 3).
(2) We present *SocIoTy*, an at-home cryptography system designed for non-expert users and their smart homes, and show how it can be used to build relevant constructions such as time-based one-time passwords [46] and self-revocable encryption [65] that can only be operated in the home context (Section 4).
(3) We implement and evaluate SocIoTy on realistic hardware, providing microbenchmarks for individual cryptographic operations on representative IoT devices and full benchmarks of an end-to-end deployment on a realistic smart home configuration (Section 5).

2 BACKGROUND

Smart homes. SocIoTy relies on a home Internet-of-Things network, or "smart home", to bootstrap at-home cryptography. IoT devices and smart home networks are proliferating rapidly [44]. In 2022, 57.5 million Americans lived in a smart home accounting for 45% of US households, and it is estimated that by 2026, more than 25% of homes worldwide will have some degree of IoT capability [63]. IoT devices range in computational capacity from extremely lightweight microcontrollers to fully-Linux-capable system boards with gigabytes of RAM. We rely on IoT devices to perform cryptographic operations and to communicate over the network in order to manifest a cryptographic scheme from the participation of otherwise logically isolated systems. As in prior work (*e.g.*, [40]), we assume a network of constrained devices (in terms of computation and power) participates in the cryptographic protocol, and in our evaluation (Section 5) model such devices to demonstrate feasibility.

Pseudorandom functions (PRFs). A pseudorandom function [27] is one which outputs values that, without knowing some key k, cannot be distinguished from random. PRFs can be used to build many other primitives in cryptography, including symmetric encryption and authentication schemes. For encryption, there exists a well-known theoretical construction that randomly chooses input for the PRF and treats the output as a one-time pad for the message. It is also possible to treat the output of a PRF as input to a key derivation function for block ciphers. Authentication is straightforward, as the input to the PRF can be the message the party would like to have verified, and the output being the tag for verification.

(Threshold) Distributed PRFs. One useful variant of a pseudorandom function is the distributed pseudorandom function (DPRF), which allow a group to *jointly* evaluate a PRF. Each party uses shares of the secret key to calculate a partial output that can later be combined to recover the full PRF output. This can be extended to the threshold case, where the computation is successful if t parties supply honest recovery values, but to any group of $t-1$ parties the PRF output appears to be uniformly random, creating a threshold DPRF (TDPRF). While TDPRFs can be constructed using generic MPC, it would be highly inefficient and take multiple rounds of communication to produce a result, both non-starters for IoT devices. We instead utilize a protocol that requires only one round of communication between evaluators and an aggregator, with no communication required between evaluators. The protocol, originally presented by Naor et al. [47], is based on the decisional Diffie-Hellman assumption and the use of random oracles.

The interface of a TDPRF consists of a tuple of algorithms (Gen, PartialEval, Recon):

- $\mathsf{Gen}(1^\lambda, k, t, n)$ produces shares of the PRF key k denoted as $k_1 \dots k_n$.
- $\mathsf{PartialEval}(k_i, x)$ uses a key share k_i on an input x to produce a partial evaluation of the PRF, y_i.
- $\mathsf{Recon}(\{y_i\}_{i\in Y})$ takes a subset of partial evaluations by users $Y \subseteq [n]$ where $|Y| \geq t$ and produces the full PRF output y.

Finally, we note that a TDPRF may have an additional efficient algorithm which takes in a fully reconstructed key k and an input x which we denote by $\mathsf{Eval}(k, x)$, allowing a TDPRF to be used as a regular PRF.

Two-factor authentication (2FA). To increase user security, online service providers have started to roll out 2FA, which requires a second form of authentication to log in to a service. The most common form of 2FA after email and SMS [19] is the time-based one-time password (time-based OTP or TOTP) [46]. Every h seconds, a user's token (*e.g.*, a smartphone app) generates an OTP. When the user wishes to authenticate, they input their username, password, and OTP. Unlike passwords, OTPs are short lived; they are only valid for the time interval h in which they are generated, and can only be used once. Users therefore authenticate with either *something they know* (a password) or *something they are* (a fingerprint or retina scan), alongside *something they have* (their token, which generates OTPs). TOTP is supported by major social media platforms [68], electronic health records systems [24], financial institutions [54], and corporations [22]

The security of TOTP relies on the underlying HMAC-based OTP algorithm (HOTP) [45], which generates OTPs using the HMAC construction [39]. The security of HOTP, in turn, relies on the assumption that HMAC is a PRF. Since adversaries without k cannot predict PRF outputs, they also cannot predict OTPs. Thus, as long as we assume that our underlying primitive (HMAC) is a PRF, then the OTPs generated by HOTP (and TOTP) are secure.

More formally, a TOTP is parameterized by h and defined by $\mathsf{TOTP}_h(k_t, z) = \mathsf{PRF}(k_t, \lfloor \frac{z}{h} \rfloor) \mod 10^6$, where z is the current timestamp and $\mod$ is the modulus operation, which is used to convert the output of PRF into a 6-digit integer. We omit h in our notation for TOTP for simplicity, and use the recommended default $h = 30$ from the TOTP RFC [46].

Compelled access. Compelled access to a software system or to data, whether by a malicious attacker or law enforcement agent, poses a serious risk to privacy and security. Compelled access can be viewed as exploiting a user's ability to authenticate, and similarly compelled decryption can be viewed as exploiting a user's ability to decrypt sensitive data. Recent work has explored the mechanisms and mitigations of compelled access and decryption in mobile devices [73], as well as defending cryptographic protocols from compelled decryption by identifying and reducing long-lived secret values [56]. BurnBox [65] attempts to address compelled decryption by putting a user's ability to decrypt their files in escrow in a safe place—specifically, by allowing for a form of secure deletion (*revocation*) which is reversible only with a secret key saved elsewhere, *e.g.*, in a vault at home.

Context-sensitive cryptography is a powerful mechanism when considering compelled access and decryption. Broadly, context-sensitive cryptography ties cryptographic operations to some notion of context, usually through some information only available in a certain place or among certain parties (*e.g.*, [61, 62, 66]). If cryptography is only possible when a secure context is established (*e.g.*, in the home), and compelled access or decryption can be expected to occur elsewhere (*e.g.*, at a border crossing or the proverbial dark alley), these risks are mitigated. Better yet, the user cannot be directly coerced (*i.e.*, via "rubber-hose" cryptanalysis) to release a key which is only accessible under a certain context.

3 DESIGNING AT-HOME CRYPTOGRAPHY

To capture the notion that some cryptographic operations should only be available within the context of the home, *i.e.*, at-home cryptography, it is necessary to modify the interface to cryptographic calls with a *context* input. This modification clearly captures generic context-sensitive cryptography, a superset of at-home cryptography. As we are only interested in this subset, we (informally) modify a cryptographic function F with input z to produce the function F_{home} as follows:

$$F_{\text{home}}(z, \textsc{context}) = \begin{cases} F(z) & \text{if } \textsc{context} = \text{home} \\ \perp & \text{otherwise} \end{cases}$$

We emphasize that this notation is informal; by making the context an input to the function, an adversarial caller could call the function with a context other than their own. Formally modeling this transformation would require limiting the caller to use their true context, perhaps by letting users make queries to a subset of functionalities, where the subset is determined by their present context. Indeed, this better matches our envisioned system, in which these oracles are realized by distributed computation on hardware segmented to only a local network (*i.e.*, the home context). In either sense, providing a formal framework for at-home cryptography is beyond the scope of this work; we will use the informal notation described above, as the intuitive meaning is clear.

We focus on at-home cryptography in this work, but this approach is general, and there is nothing preventing the above definition from being applied to other contexts. For example, one could envision an "at-work" cryptography system for employees performing sensitive tasks in an industrial IoT setting.

We note that there are times when context is also (practically speaking) location-bound. For example, if the devices that define the home context are difficult or expensive to move (*e.g.*, a smart ceiling fan, a smart oven, or a smart irrigation system), at-home cryptography could also realize a limited form of *location-sensitive* cryptography. We explore this idea further in Section 4.4. Of course, if the user desires to change homes, the at-home context should be able to move with the user; the user has a new definition of "home", and at-home cryptography should reflect that. As such, we consider the at-home context to be semi-permanent.

3.1 Case Studies

To make the envisioned usage of at-home cryptography clear, we briefly present several concrete use cases. While not true anecdotes, these motivating examples are rooted in real-world trends and contextualize the technical considerations that must go into designing our at-home cryptography solution, SocIoTy.

Use Case 1: The Remote Worker. Consider a user that recently accepted a job offer from a prominent law firm as a legal aid, where they will work as a remote-only employee; this kind of remote-only work has been on the rise since the COVID-19 pandemic [49], and some anticipate that many of these jobs will remain fully-remote permanently [55]. To access the sensitive legal documents required to do their job, the user connects to the law firm's network over a VPN. To authenticate to the VPN, the user enters codes generated by a 2FA app on their company-managed smartphone. Company policy requires that the user should only connect to the VPN when within their home, owing to the sensitive nature of the company's documents. However, the user has no way of ensuring that they meet that policy if their smartphone is lost or stolen.

Use Case 2: The Outpatient. Consider a user who has end-stage renal disease, and requires active management through dialysis. Instead of remaining in the hospital, the user owns a dialysis machine at home, which are increasingly common [41]. The user regularly meets with their doctor to discuss their condition. On days they are not able to visit their doctor for a check-up, they set up a telemedicine appointment from home. Based on the check-up, the user's doctor is able to remotely configure the dialysis machine over the Internet. The user has an app they use to connect to hospital's electronic medical records system, but is nervous about their health data leaking when they leave the house.

Use Case 3: The Foreign Correspondent. Consider a user who is an investigative journalist that frequently travels to war-torn, authoritarian countries as part of their reporting duties. During such trips, the user keeps detailed notes, initial research, article drafts, and the identities of sources on their smartphone. To ensure that this information is not lost if their phone is lost, they back up these files to cloud storage services; due to the sensitive nature of these documents, they keep them encrypted while in cloud storage and keep decryption keys on their smartphone. While traveling, the user is often stopped and searched by local law enforcement (either at border crossings or during routine encounters on the street); such stops are common in countries with repressive regimes, and border officers are known to extract data from smartphones at border crossings [73]. The user has heard of next-generation cryptographic systems designed to let them temporarily revoke access to their sensitive documents until they return to a secure location (*e.g.*, [65]), but they lack the dedicated, stationary hardware those systems require.

3.2 Design Goals

With these use cases in mind, we now discuss the goals and considerations for a realization of an at-home cryptography system.

Functionality. All of the use cases in Section 3.1 require limiting execution of cryptographic functions to the home. One way of implementing this constraint is to only hold the secrets at home, so the required key material is unavailable under any other context. This would allow all three of our envisioned users to opt-in to limiting access; each envisioned user either does not require access on-the-go, or would like to ensure it is not possible.

We require support for both authentication (use cases 1 and 2) and encryption (use case 3). An authentication primitive means that we can use the home as a second factor for 2FA—*somewhere you are* in addition to the typical *something you know, have,* or *are* triad. An encryption primitive would allow users to secure files such that they can only be decrypted when the user is at home, suitable for privacy systems that require a digital safe [65] to recover files after a threat has passed and they have returned home. Note that it is possible to accomplish this task robustly while still storing encrypted files in the cloud—even if the encrypted files are available globally, the plaintext documents are context sensitive.

Deployability. Prior work on context sensitivity for cryptographic operations [6, 14, 43, 60–62, 66] has not been deployed in practice due to its reliance on specific hardware to provide security properties or non-standard adversarial models. Therefore, we aim to use *existing* devices to build the context: namely, the IoT devices of the user's smart home. Since we are re-using devices, we must ensure that our solution does not require intensive or long-running computations. Similarly, we also prefer protocols that have as few rounds of communication as possible (or, ideally, are non-interactive) and linear in communication complexity.

In essence, the IoT devices should achieve their cryptographic task quickly and return to their primary functionality in the home. From the user's perspective, the only change is that the context matters for the task at hand; the rest of the interface for cryptography should be the same, and be fast enough that the user does not notice any latency.

We note that this does not preclude a more powerful device from being involved. The IoT devices can operate a lightweight part of the computation; then, *another* device—a smartphone or tablet—performs the more heavyweight computation. This other device and its interface can be the same as what would be used in a more traditional, non-context-sensitive cryptographic solution (*e.g.*, a 2FA app on a smartphone), abstracting away the new at-home cryptography system.

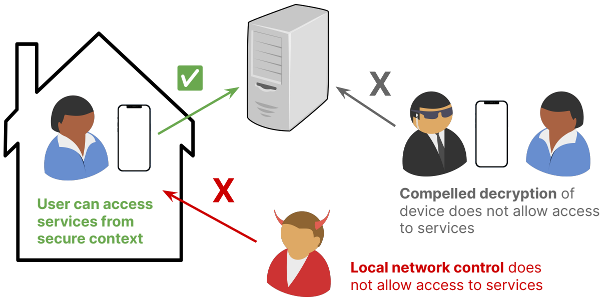

User can access services from secure context
X
X
Compelled decryption of device does not allow access to services
Local network control does not allow access to services

Figure 2: An overview of our threat model for SocIoTy.

Security. An at-home cryptography system must be secure in the context of adversaries that are able to corrupt and control the smart home's devices, and those that are able to compel access to the user's secrets outside of the home. We more concretely define our threat model in Section 3.3.

The smart home setting introduces particular challenges not captured by traditional models. For example, family members and roommates can also share the space, perhaps with their own IoT devices. Additionally, each member of the household may have several different at-home services they wish to use. Any solution must be therefore secure in the presence of several *other* users and multiple *different* services.

We emphasise that our intention is to allow for users to *opt-in* to this extra layer of protection for the selected services that make sense for them (or the organizations of which they are a part). We target users who are explicitly concerned with the risks associated with ubiquitous access to *all* online services and files, and want to choose which services they can access on-the-go versus when in the home context. Critically, these choices are highly specific to each user. In use case 2, for instance, the user's fixed medical devices should *only* be able to communicate at home; any communication outside the home would likely be an error. Moreover, the choice to add context sensitivity to particular services might also change over time, based on what the user plans to do when they leave their home and the specific threats that they might expect along their journey. For example, in use case 3, the user might want to add an additional layer of security to their sensitive services only when planning to cross international borders, even if they do not context-bind access to these services in their daily life.

3.3 Threat Model

Since our system makes use of multiple devices and a variety of scenarios, it is important that our threat model systematically considers all of these components. We model around a setting where the user has IoT devices in their smart home, as well as a powerful mobile device (a smartphone or tablet) that can communicate with the IoT devices and can be involved in a setup procedure that authorizes it to participate in the protocol. We will refer to this device as the *authorized device.* To successfully tie cryptographic services to the home, the system must be designed in such a way that a cryptographic operation cannot succeed if this authorized device is not also at home and participating in the protocol.

Concretely, we demonstrate this by showing that the protocol must be secure against the following types of adversaries:

Compelled access adversaries. We consider adversaries that can obtain access to the authorized device when it is not physically present in the home [73]. These adversaries can extract all of the secrets from the authorized device. For example, consider a border control officer that compels decryption of the user's authorized device while the user is crossing the border [65]. While they now have access to any secrets on the authorized device, they should not be able to successfully authenticate or decrypt as they are not physically in the home. We note that this threat model is stronger than that of many secure protocols, which assume a malicious network but a trusted, secure end-user device.

Local network adversaries. We consider adversaries present on the local network. This can occur through compromising any number of IoT devices on the network. This is a natural assumption as IoT devices have a history of vulnerabilities [23, 58]. The adversary can be remote, or have physical access to the devices. The latter models threats from other residents of the smart home, such as a malicious roommate or house-guest. In either case, the local network adversary will also be able to see all of the traffic over the LAN between the devices and the authorized device. They would also have the ability to use this access to perform denial-of-service. Despite all this, as long as the user's authorized device remains secure, the local network adversary should not be able to successfully execute the protocol.

These two adversaries represent the primary ways that a malicious party would try to undermine an at-home cryptographic system. Building a system robust against these two adversaries ensures that in-home compromise of the IoT devices or out-of-home compromise of the user's authorized device does not violate the context-sensitive property of the system. We design for this threat model in a modular way, demonstrating the security of our system against each adversary independently. We assume that these adversaries do not collude, as compelled access adversaries are not assumed to have the capability to access personal devices besides those physically available to them [56, 65, 73]. However, in Section 4.3 we describe some extensions that would allow for our system to handle colluding adversaries as well.

4 SOCIOTY

We are now ready to describe SocIoTy, our at-home cryptographic solution.

4.1 Preliminaries

We discuss SocIoTy in terms of its components:

- *Authorized device/authorized smartphone*: This device has reasonably good computational power and is carried by the user. We assume the authorized device is honest while within the home, but might be corrupted (*e.g.*, stolen or forcibly removed from the user) upon leaving the home. We assume the device supports *effaceable storage*, *i.e.*, allows for secure deletion of cryptographic secrets. Such functionality is common on modern smartphones [73].
- *Remote service*: The remote service is an Internet-accessible service with which the user wishes to interact through their at-home cryptography. In the authentication case, this is a service requiring login with 2FA enabled. In the encryption case, this is a cloud storage endpoint.
- *IoT devices*: The user selects the IoT devices from their smart home with sufficient hardware and network capabilities to execute the SocIoTy protocol, which may include smaller, microcontroller-class devices.

Choosing the correct cryptographic primitive. For both at-home authentication and encryption, we need to tie cryptographic operations to a particular context. One natural way to do this is to have IoT devices use a generic MPC protocol to perform both encryption and authentication, where the respective keys have been secret shared among all parties and the output is given directly to the smartphone.

Unfortunately, generic MPC is too inefficient for our setting, involving multiple rounds of communication and expensive computation operations [36, 37, 67]. Particularly in IoT environments, where even RAM is significantly constrained, we cannot use many standard tricks to improve performance and even hundreds of milliseconds per circuit layer may introduce unacceptable latency. Additionally, the comparatively high resource requirements of an MPC protocol may interfere with the normal operation of the smart home. Waiting for the interactive execution of each round could delay the processing of incoming IoT events, and this delay could be exacerbated by the limited multitasking capabilities of IoT devices. We further explore MPC as a primitive for SocIoTy in Appendix B, where we highlight the slow execution times of an MPC-based protocol on IoT devices.

We would instead prefer to have the smart home implement a single cryptographic primitive that is well suited for use in a wide range of applications. One primitive that could work is a PRF, which has standard transformations to both symmetric encryption schemes and MACs. The distributed version of a PRF that is most applicable in our setting is a TDPRF [47]. As discussed in Section 2, a TDPRF allows $\geq t$ parties to compute partial evaluations of the PRF that can later be combined to recover the full PRF output, but to any group of $< t$ parties, the output of the TDPRF is indistinguishable from random. This helps with both security and availability; not every IoT device needs to be online to evaluate the TDPRF, but any adversary that only compromises $< t$ devices cannot recover the correct output of the TDPRF on any point that they have not already seen.

Layering security. While a TDPRF achieves some security against an adversary corrupting $< t$ parties, we would also like to handle the case where an adversary corrupts over this threshold, potentially even up to all the IoT devices in the home. To protect against such adversaries, the phone will also contribute to constructing correct output. In short, we will construct a new PRF P' from the proposed TDPRF of the smart home and a PRF P, with the same co-domain as the TDPRF. If the composition of the outputs of the TDPRF and P is pseudorandom, even when either of the TDPRF key or the key for P is leaked (and the smartphone is the only party who holds the key for P) the output of P' will appear pseudorandom to all adversaries covered in our model. This layering will also be more

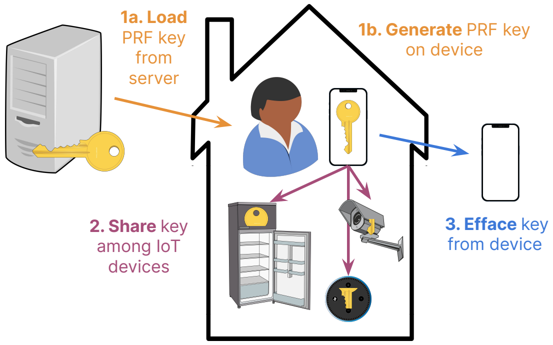

1a. Load PRF key from server
1b. Generate PRF key on device
2. Share key among IoT devices
3. Efface key from device

Figure 3: The setup workflow of SocIoTy.

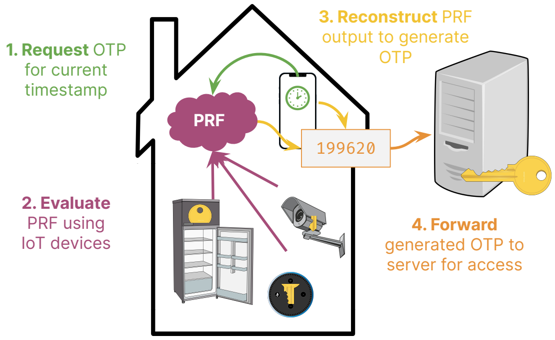

1. Request OTP for current timestamp
3. Reconstruct PRF output to generate OTP
PRF
199620
2. Evaluate PRF using IoT devices
4. Forward generated OTP to server for access

Figure 4: The authentication workflow of SocIoTy.

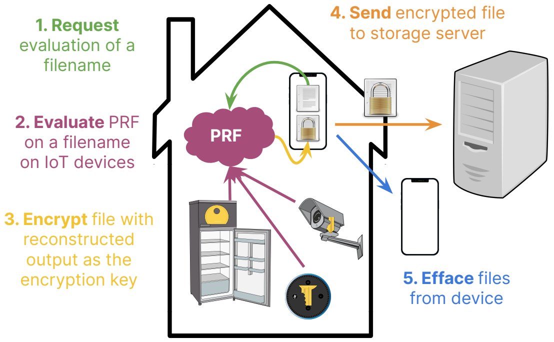

1. Request evaluation of a filename
4. Send encrypted file to storage server
PRF
2. Evaluate PRF on a filename on IoT devices
3. Encrypt file with reconstructed output as the encryption key
5. Efface files from device

Figure 5: The encryption workflow of SocIoTy.

practically efficient to compute than any generic solution that only protects against a limited number of IoT device corruptions.

4.2 Protocol Description

We now briefly describe the normal operation of SocIoTy at a high level before describing the protocol in depth. When an authorized smartphone wants to register a new service with the smart home, it first generates the key material needed for itself and the home

Algorithm 1: SocIoTy authentication

```
Input: $k_p$ the key of the smartphone, $\delta$ a counter value
       derived from a timestamp
Output: TOTP token
Request smart home devices invoke PartialEval on $\delta$ and
  receive $\{y_i\}_{i \in T}$ where $T \subseteq [n]$, $|T| \geq t$
$y \leftarrow$ TDPRF.Recon$(\{y_i\}_{i \in T})$
$z = y +$ PRF.Eval$(k_p, \delta)$
Output $z \pmod{10^6}$
```

using a setup algorithm (Figure 3). It gives the correct key shares to all the devices in the smart home and securely deletes them from its memory. When the smartphone later uses the service, it broadcasts over the LAN a request for a TDPRF evaluation. The phone waits until it receives at least t evaluation responses from the IoT devices before attempting reconstruction. Once reconstruction is completed, depending on whether the application is authentication (Figure 4) or encryption (Figure 5), the phone takes a series of actions. Any sensitive information is securely erased from the phone after the operation completes. What follows is a complete description of the setup, authentication and encryption algorithms for SocIoTy.

Setup. Let n be the number of smart devices a user owns and let t be a fixed number, equal to the number of devices expected to be online at any given point in time. The setup procedure is designed to produce two keys: one for the smartphone denoted by k_p and one split among the networked IoT devices denoted by k. In the case of authentication, the keys k_p and k will be provided by the remote service the phone is authenticating to. For encryption, k_p and k should be generated by the smartphone. The phone uses the TDPRF.Gen algorithm to share the key k as $k_1 \dots k_n$. The key share k_i is given to device i. The phone then stores k_p and after sharing the shares of k, securely deletes all key material related to k. When a new IoT device is bought or sold from the smart home, the phone repeats this procedure, replacing k_p and k with new keys. Figure 3 illustrates the setup process. We assume that in the case of authentication, the remote service provides a mechanism by which the symmetric TOTP key can be updated.

We note that the setup algorithm should be run for each remote service for authentication, as each remote service would require its own key. Similarly, we recommend running the setup algorithm for each set of files that are to be encrypted. This way, all SocIoTy applications have their own (k_p, k) pair.

Also, to ensure that all secrets are initialized without adversarial interference, we require the setup process to be over a secure, authenticated point-to-point channel. This channel can be instantiated over TLS with mutual authentication (bootstrapped via, *e.g.*, QR codes). After this setup procedure, however, SocIoTy does not have this secure channel requirement for communication, as the dual-layered PRF prevents an adversary from getting total PRF output even if they obtain k. We discuss this further in Section 4.3.

Authentication. For authentication, the smartphone calculates a counter δ based on the current timestamp. It then sends an authentication request to all devices within the smart home. Each device with available bandwidth runs TDPRF.PartialEval(k_i, δ) to get y_i and sends the resulting y_i to the phone. Once the phone

Algorithm 2: SocIoTy encryption

```
Input: $k_p$ the key of the smartphone, $m$ the name of the file,
        $f$ the content of the file itself
Output: Encrypted file $c$
Request smart home devices invoke $\mathsf{PartialEval}$ on $m$ and
  receive $\{y_i\}_{i \in T}$ where $T \subseteq [n]$, $|T| \geq t$
$y \leftarrow \mathsf{TDPRF.Recon}(\{y_i\}_{i \in T})$
$z = y + \mathsf{PRF.Eval}(k_p, m)$
$K \leftarrow \mathsf{KDF.Derive}(z)$
$c \leftarrow \mathsf{AE.Encrypt}(K, f)$
Securely delete $K$
Output $c$
```

has received t partial evaluations it recovers the PRF output and calculates its own PRF value. The two are then combined[2] and the output is truncated to 6 digits, which are displayed to the smartphone user. The remote service can use k_p and k, along with the PRF.Eval and TDPRF.Eval algorithms, to check for correctness. An overview diagram is provided in Figure 4, with a more specific description in Algorithm 1.

Encryption. To encrypt and decrypt sensitive files, the smartphone first makes a request for a TDPRF evaluation on a filename m. The smart home devices conduct a partial evaluation as $\mathsf{TDPRF.PartialEval}(k_i, m)$. The output of these evaluations is then given to the smartphone. The phone can then reconstruct the TDPRF output before combining it with the output of its own PRF evaluation on m using key k_p. The resulting output is then used as an entropy source for a key derivation function, KDF [38]. The value returned by KDF is a pseudorandom key[3] which can then be used in an authenticated encryption scheme AE to either encrypt or decrypt the file while providing strong confidentiality and integrity. After the operation is completed, the phone securely deletes the reconstructed key and potentially the plaintext file. This workflow is depicted in Figure 5 and described in Algorithm 2.

4.3 Security Analysis

We now give a justification of security for our construction against the relevant adversaries. Recall that we are concerned with two types of attackers (1) a compelled access adversary who may compromise the phone while it is abroad, but does not simultaneously have access to any device in the smart home and (2) a local network adversary that has direct physical access to IoT devices and any traffic over the LAN but cannot compromise the smart phone. We note that in the multi-user setting, other users are equivalent to adversary 2. To give a brief summary, security holds because of how the PRF and TDPRF are composed. Even if an adversary has access to one of k_p (adversary 1) or k (adversary 2), the total output retains PRF security and is indistinguishable from uniform. This is because the evaluations of the smart home TDPRF and smartphone PRF are *additively composed*. If one of these keys is unknown and drawn from a uniform distribution, then the output of the (TD)PRF evaluation with that key will be unknown. Then, the sum of this unknown value with the other (known) PRF evaluation is still uniform. This means an adversary has no chance better than random of guessing either the TOTP value or the key used to encrypt files.

Extensions. We now discuss some special considerations for other types of network attacks and more powerful adversaries.

A local network adversary in practice has some slightly stronger adversarial capabilities, due to its ability to modify traffic and directly interact with the smart home. SocIoTy does not necessarily require authentication and encryption of home requests, as the security of the system relies on the dual-layered PRF. Thus, even if the adversary attempts to relay into the smart home and interact with it remotely, or replay a prior request to the devices, they will not be able to obtain the the final TOTP as they do not have k_p (which is on the user's smartphone). However, without guaranteeing the authenticity of the requests, we do open up users to denial-of-service attacks on each of the relevant services, as an adversary could interfere with partial evaluations[4]. In the case of authentication, such an attack can only temporarily prevent correct functioning of the system. Assuming that either the AE is key-robust or the KDF acts as a random oracle prevents related-key attacks from enabling plaintext file recovery. When initializing a user account and encrypting files, denial of service attacks may be conducted which have irrecoverable effects, leading to a breakdown of system properties (*e.g.*, encrypting a file with a corrupted key and then deleting the plaintext).

So, if protection against these attacks is required, we recommend adding authenticity checks to values sent by IoT devices. The secrets required for these checks can be generated and shared during setup. In our implementation, we use the recently standardized Ascon lightweight AEAD scheme [48] for communication between parties to maintain security against these types of attacks. We evaluate the performance of Ascon in Section 5.

Next, we consider a more powerful access adversary who can gain control of both the phone and even one IoT device on the home network to be out of scope. We believe, for most use cases, this is a realistic assumption: even gaining the public-facing IP address of devices on the network is not something a foreign nation can do easily, without help from the user's local ISP. For those highly-targeted users for whom such an adversary could be realistic, though, turning off the smart home entirely when they leave the house will prevent this attack, giving the user the same security guarantees as an offline solution.

Finally, we consider an adversary that first compels access to the user device and then later attempts to compromise the rest of the system to be also out of scope. We note that prior work on compelled access [65] similarly does not provide security against attacks that occur after the access ends. To achieve post-compromise security in this setting, our system could be extended to perform a full key rotation upon returning home after a compelled access event. This process would involve regenerating k_p and k for all authentication services and all encrypted files.

[2] We combine these two values through addition, which occurs in the codomain of the underlying PRFs.

[3] We assume that KDF acts as a random oracle, which defends our construction from related-key-style statistical attacks on the encryption key.

[4] We note that, without authentication, a DoS is possible by anyone with at least some local network access; it does not require the ability to corrupt an IoT device.

4.4 Deployment Flexibility

SocIoTy's design allows for significant flexibility when deploying on a smart home. We discuss these considerations below.

Devices to use. We envision SocIoTy as running on essentially any IoT device that has some form of networking capability, as the number and types of device vary from smart home to smart home. Users should try to use their more powerful devices to increase performance, but we believe this is not strictly necessary. We evaluate these performance claims in Section 5, using a wide range of devices to benchmark the SocIoTy protocol.

Multi-user smart homes. SocIoTy can support multiple users, each with their own services. Each device i holds a separate key k_i (for PartialEval) for each pair (u, s) describing a user u and service s. The wrong user u' cannot authenticate to s as u because they do not have the key k_p on u's smartphone. Thus, SocIoTy supports as many users and services as there is space for keys on the IoT devices. Similarly, SocIoTy also supports *multi-owner* setups, where the devices are not all owned by a single user. This is common in smart home settings, as devices can belong to roommates, landlords, or caretakers, to name a few. If all device owners cooperate, SocIoTy proceeds as normal. If owners deviate from the protocol, the worst that can happen is denial-of-service—not a security break.

Network structure. Our design does not require a specific structure of the smart home network. Traditionally, protocols are designed point-to-point, where each device is able to directly communicate with each other device. For some smart homes, computation is handled through a hub, which acts as an intermediary for messages to and from the smart home devices. Hubs are particularly used for low-resource devices. SocIoTy is able to handle this case, which we investigate end-to-end in Section 5.3.

SocIoTy makes no liveness assumptions on the whole network. Other approaches, like generic multi-party computation [11, 18, 28, 70], would require all of the IoT devices to communicate with each other during the whole protocol. SocIoTy only needs each node to be active for one PartialEval. So a device can respond to a request, and go back to attending to its primary task (or return to sleep), without waiting for all of the other nodes to respond or for the final reconstruction to occur. Moreover, because our cryptographic protocol only requires one round of communication, we can also tolerate networks with very low available bandwidth.

Server interface. SocIoTy meets our deployment goal of not requiring changes to the user interface, but we briefly discuss how SocIoTy impacts the *remote service.* When applying SocIoTy to encryption, the cloud server that provides storage does not change its interface. From its perspective, the user is still uploading a file: a SocIoTy-encrypted blob rather than a cleartext one. The cloud service stores it as it would any other file.

For authentication, however, the situation is different. The TOTP standard [46] recommends HMAC-SHA-1 as the underlying PRF. Our construction is not backwards-compatible with HMAC-SHA-1 in implementation, but the interface is the same: a call to TOTP returns a one-time password. Rather than using HMAC-SHA-1, a call to TOTP(k_t, ts) in SocIoTy would instead invoke Algorithm 1, with the server keeping $k_t = (k_p, k)$.

We argue that this change is minimal, as the TOTP standard has a high level of abstraction [46]. Moreover, services are incentivized to make this change, as the additional security and flexibility of SocIoTy is a marketable benefit.

From context to location. So far, we have defined SocIoTy in terms of *context*, where the context is the home, or more precisely, the presence of smart home devices with specific cryptographic material in the home. Some users may wish to go the extra step and attain true *location sensitivity*, *i.e.*, binding their computation to their physical house, rather than just the context of the devices inside of it. Location sensitivity is a physical attribute, and as such needs physical-level steps to integrate with SocIoTy's context sensitivity. The most straightforward way would be to physically bind devices to the home, bolting down SocIoTy devices into walls or using devices for SocIoTy that are cumbersome to move (*e.g.*, a smart fridge). Another option is to enforce location during communication, by using low-range technologies such as NFC for communication or by validating certain radio attributes at the PHY layer (*e.g.*, [66]). Location can also be tied to some sort of user interaction in which presence is required, such as a button press or voice command. We leave integration of specific location-based security controls as future work.

4.5 Instantiating the TDPRF

We must instantiate the TDPRF underlying SocIoTy's operations to deploy our solution in practice. We employ the decisional Diffie-Hellman-based construction first proposed by Naor et al. [47] for our TDPRF. We choose elliptic curve groups for the underlying operations because of their efficiency in implementation. As is common when discussing elliptic curves, we use additive notation for group operations. Let G be a generator of an elliptic curve subgroup $S \subseteq E(\mathbb{F}_q)$ of prime order p and $\mathcal{H} : \{0,1\}^\lambda \to S$ some hash function modeled as a random oracle hashing λ-bit strings onto S. We describe below the algorithms for our TDPRF Gen, PartialEval, and Recon, as well as the extra algorithm Eval (useful for a server implementation):

- Gen$(1^\lambda, k, t, n)$: Sample a random polynomial f of degree $t-1$ by uniformly sampling its coefficients from $\mathbb{Z}_p$, subject to the constraint that $f(0) = k$. The output party shares are the scalars $k_1 = f(1), k_2 = f(2), \ldots, k_n = f(n)$.
- PartialEval(k_i, x): Hash the input x using $\mathcal{H}$ onto a point P along the elliptic curve. Then, the output is simply $y_i \leftarrow k_i P$: scalar multiplication of the *key share* to the input point.
- Recon$(\{y_i\}_{i \in Y})$: Let $\alpha_1 \ldots \alpha_t$ be the identities of the parties providing points $y_1 \ldots y_t$ to Recon. Consider the following function, defined $\forall i \in [t]$:

$$L_i(x) = \prod_{\forall j \neq i, j \in [t]} \frac{x - \alpha_j}{\alpha_i - \alpha_j}$$

It is well known that given t points along a polynomial f, evaluation can be done at any point α as $f(\alpha) = \sum_{i=0}^{t} f(\alpha_i) \cdot L_i(\alpha)$. Given these points "in the exponent" it is possible to recover a "in the exponent". To be precise, we can recover

Table 1: Hardware specifications of test devices as well as examples of comparable IoT devices.

Test Device	CPU	RAM	Comparable IoT Device
RPi 3B+	ARM Cortex-A53	1 GiB	Apple TV HD [7]
RPi 2B	ARM Cortex-A7	1 GiB	Amazon Echo Dot (3rd Gen) [12]
RPi Zero W	ARM1176JZF-S	512 MiB	Google Nest Thermostat E [42]
ESP32	Xtensa LX6	320 KiB	Belkin WeMo Light Switch [64]

the PRF output as:

$$y = \sum_{i=1}^{t} L_i(0) \cdot y_i = \Big[\sum_{i=1}^{t} L_i(0) \cdot f(\alpha_i) \Big] P = kP$$

- Eval(k, x): Hash the input x using $\mathcal{H}$ onto a point P along the elliptic curve. Then, the output is $y \longleftarrow kP$: scalar multiplication of the *reconstructed key* to the input point.

Note that the hashing of the input x is important, as the output of this TDPRF is uniform only if its input is also uniform. If we model this hash function as a random oracle, security holds [47].

5 EVALUATION

We now demonstrate the feasibility of our constructions on real IoT hardware.

Implementation. We implement SocIoTy in Rust due to its memory safety guarantees as well as its good platform support for IoT architectures. Additionally, we use the Curve25519 as our elliptic curve and Ascon, the winner of the NIST lightweight cryptography competition [48], for authenticated encryption in our implementation. While the dual-layer PRF allows for security to hold during evaluation without the need of authenticated encryption, we include it in our implementation to add security against network tampering (as discussed in Section 4.3) and evaluate its overhead. We have open-sourced all of our SocIoTy software and benchmarks for public use and review[5].

Devices. We wish to understand how SocIoTy runs on a variety of devices. To this end, we performed our benchmarks on the following devices: 6 Raspberry Pi (RPi) Model 3B+ single-board computers (SBCs), 3 RPi Model 2B SBCs, 3 RPi Zero W SBCs, and 5 ESP32 microcontrollers. Raspberry Pis are increasingly being used a benchmarking platforms to simulate smart home devices in lieu of commercial devices; IoT device vendors do not support running arbitrary software for security reasons, limiting the ability to use them for development. Recent generations of Raspberry Pis have been increasing in computing power with specifications of up to 8GB of memory. Thus, we used both lower-end Raspberry Pis and smaller microcontrollers as representative devices to better simulate a network of heterogeneous IoT devices. Table 1 maps our test bed devices to comparable smart home devices.

5.1 Microbenchmarks

We first begin by presenting microbenchmark results for the algorithms of a TDPRF: (Gen, PartialEval, Recon). All of our selected devices are capable of computing all three of these algorithms.

A core goal of SocIoTy is to re-use existing smart home hardware to provide cryptographic services. We primarily study the execution time of SocIoTy primitives in our simulated smart home, identifying if the additional latency of running SocIoTy would hinder the normal operation of the device. Through these microbenchmarks, we aim to investigate if adding SocIoTy software would necessitate hardware changes on the IoT devices.

Table 2: Average runtimes for an evaluation of PartialEval**, both without and with authenticated encryption** (AE)**. All times are in milliseconds.**

Experiment	RPi 3B+	RPi 2B	RPi Zero	ESP32
PartialEval	1.34	2.19	2.90	43.68
PartialEval (AE)	1.53	2.43	3.28	47.22

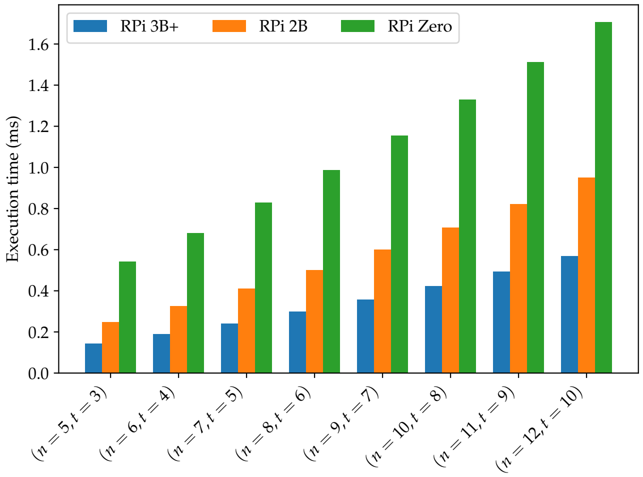


Figure 6: Microbenchmarks for Gen **on different test devices over varying configurations of total number of parties** n **and reconstruction threshold** t**.**

PartialEval. Because PartialEval will be conducted on resource-constrained IoT devices, microbenchmarks for it are very informative. As discussed in Section 4.5, each PartialEval in our implementation is one elliptic curve multiplication. We evaluate the performance of PartialEval, and of PartialEval followed by an authenticated encryption of the result using Ascon (denoted AE). The Raspberry Pis performed each task 100,000 times, and the ESP32 performed each 1,000 times, with the results in Table 2. The Raspberry Pis complete the task very quickly—less than 5 milliseconds on average. The sub-50ms average time on the ESP32s is also very promising; while an order of magnitude slower than the Raspberry Pis, this result shows that adding SocIoTy on even highly constrained devices will not induce noticeable latency. We also note that the overhead of authenticated encryption is minimal, even on the ESP32. As such, for the rest of our benchmarks, we have all nodes use PartialEval with Ascon.

Gen and Recon. We also perform microbenchmarks on Gen and Recon, and present our results in Figures 6 and 7. Each Raspberry Pi once again ran each task 100,000 times, with varying configurations of the total number of parties n and the threshold required to reconstruct t. Gen in our implementation only samples random values for keys. So, while the time to run does scale with each (n, t)

[5] Available at https://github.com/tusharjois/socioty.

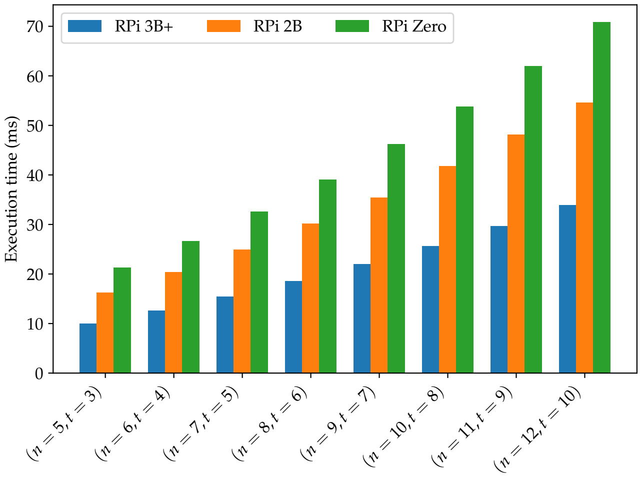


Figure 7: Microbenchmarks for Recon **on different test devices over varying configurations of total number of parties n and reconstruction threshold t.**

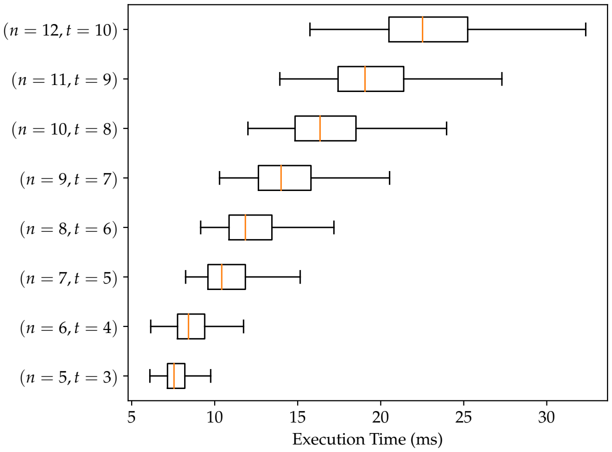


Figure 8: Protocol execution time over CoAP for varying configurations of total number of parties n and reconstruction threshold t.

pair, the operation only requires several hundred microseconds (μs) on average. Recon takes longer, likely owing to the multiple elliptic curve operations required to interpolate the partial evaluations and recover the PRF output. It similarly increases as the network grows, but even in a 12-device network, it takes only around 70 milliseconds on a Raspberry Pi Zero.

We believe that these microbenchmarks represent an upper bound on the execution time; as discussed in Section 4.1, we expect users to use their smartphone as the authorized device for Gen and Recon, and modern smartphones have much better processors than the ARM1176JZF-S found in the Pi Zero. While we do not envision users generating and recovering on even smaller, microcontroller-class devices, for completeness we evaluated how Gen and Recon fare on the ESP32 for different configurations of (n, t). These results can be found in Appendix A.

Resource requirements. IoT devices have limited resources, so we also evaluate how much storage and compute our implementation requires. Even in our smallest device class, the ESP32, we only use 40% of the total flash storage and one core for our binary and the runtime required on-chip. Many consumer home IoT devices have specifications that far exceed these requirements. Additionally, our current implementation is unoptimized research code, and further improvements could reduce the size of the binary further. This result nonetheless establishes a relative floor on the level of IoT device that would be necessary for our specific implementation, as well as shows that hardware modifications are unnecessary to support SocIoTy software.

5.2 Scalability Benchmarks

Our next set of experiments measures how the execution time of the evaluation of SocIoTy's TDPRF scales once communication between devices is involved. We set up two types of nodes, a request node and n evaluation nodes. The evaluation nodes are the Raspberry Pis: 6 RPi 3B+s (used for all benchmarks), 3 RPi 2Bs (used for $n \geq 7$), and 3 RPi Zeros (used for $n \geq 10$). In each run, the request node connects to all n evaluation nodes and makes a request for a timestamp δ. Each evaluation node then responds with the (authenticated-encrypted) PartialEval for δ, and the request node performs Recon once it has received (and decrypted) t responses. All of the nodes are on the same Wi-Fi network, and communication occurs over the Constrained Application Protocol (CoAP) [71], a popular point-to-point protocol in IoT.

We perform 1,000 of these runs for varying configurations of (n, t), and plot our results in Figure 8. Benchmark results for additional variations of (n, t) can be found in Appendix A.

Clearly, as the required threshold to reconstruct t increases, the execution time increases: more responses need to arrive. Moreover, we see a relatively large jump in average execution time at $n = 7$ and $n = 10$, likely because of the involvement of the less-powerful Pi 2Bs and Pi Zeros at each step. We note that waiting for the first t responses is biased towards the fastest devices, but this bias does not impact security. SocIoTy does not require entropy from the devices during PRF evaluation; rather, all of the entropy required is provided by the smartphone to generate the keys k_p and k during setup (see Section 4.2). We could extend the system to wait for a specific number of responses from different device types, which would potentially make the system more robust against the compromise of a class of devices (like a vulnerability affecting a brand of light switch). This, in turn, would allow users to weigh trust more granularly in their smart home.

Regardless, even in a relatively large configuration like $(n = 12, t = 10)$, each full run takes less than 25 milliseconds on average. Thus, SocIoTy TDPRF evaluations are able to scale well as the smart home network adds devices.

5.3 End-to-End Deployment

We now perform an end-to-end deployment of SocIoTy. We focus on the authentication process depicted in Section 4.2. The results for

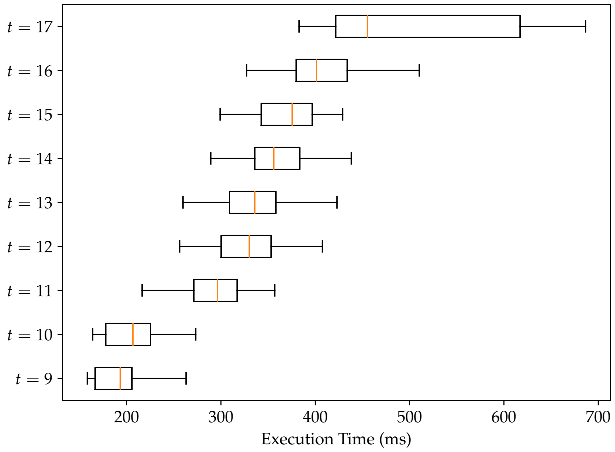


Figure 9: End-to-end OTP generation time using our iOS app for varying configurations of total number of parties n and reconstruction threshold t.

authentication will be applicable to encryption as well, as the core of the two algorithms is the same. The only difference is the actual authenticated encryption of a file, which has minimal overhead.

An end-to-end authentication system must support the generation of 2FA OTPs. So we built a smartphone app, based on an open-source implementation [9], with the same interface as common 2FA apps. Our app performs all of the steps in the TDPRF evaluation—making the PartialEval requests and Reconing the responses—and takes the additional step of converting the output of Recon into a six digit OTP and displaying it to the user. A screenshot of our app can be found in Appendix A.

As discussed in Section 4.4, smaller single purpose devices may not directly connect to the Internet or other devices found within the smart home, but rather connect to a central, more-powerful hub. This hub coordinates the flow of data of each device to and from parts of the smart home or Internet. A commonly used IoT protocol for this is MQTT, in which devices *subscribe* to *topics* to receive information and *publish* to them to send information, while a central broker sends published data to subscribers.

Keeping this in mind, we construct the following testbed to perform our end-to-end experiments. Our simulated smart home consists of the 12 Raspberry Pis from our experiment in Section 5.2, as well as 5 ESP32 microcontrollers—representing the class of devices that use lightweight IoT protocols like MQTT due to its hub architecture—for total of 17 evaluation nodes, all connected to the same Wi-Fi network. A 2018 iPhone X is used to run our smartphone app. We use a standard Ubuntu 21.04 server running on the same LAN as the MQTT broker.

Every h seconds (represented by a full progress circle in Figure 10), the iOS app generates a new TOTP by doing the following:

(1) The app calculates the TOTP counter value $\delta = \lfloor \frac{z}{h} \rfloor$ based on the current timestamp z.
(2) The app connects to the MQTT broker, subscribes to the MQTT topic `socioty/tdprf/`δ, and publishes δ to the topic `socioty/tdprf` to the broker.
(3) Each node i is subscribed to `socioty/tdprf`, and receives δ from the broker.
(4) Each node i then computes $y_i \leftarrow \mathsf{PartialEval}(k_i, \delta)$, and publishes it to `socioty/tdprf/`δ.
(5) Once the app has t responses, it performs the remainder of Algorithm 1, reconstructing the output and displaying the new TOTP.

We consider the above steps one run, and we perform 100 runs, varying the threshold t while leaving the number of total devices fixed as $n = 17$. We present our results of our end-to-end deployment benchmarks in Figure 9. We see that average execution times range from under 200ms at a majority threshold $t = 9$ to under 500ms when all devices are involved at $t = 17$. Similarly to our results in Section 5.2, we see a sharp increase in execution times as we rely more on weaker devices to provide their responses. The large spread at $t = 17$ is likely due to the app waiting for a straggler device that receives the request last and computes a response last; after all, an n-of-n system will be as fast as its slowest component.

Our experiments show that we are able to request and reconstruct the OTP well within the default lifetime of the TOTP, which is $h = 30$ seconds [46]. For a threshold set to a simple majority of devices the response is quick, accounting for less than 1% of the TOTP lifetime. We find these results demonstrate the practicality of our system to be used seamlessly as a TOTP generator.

Barriers to widespread deployment. The above result demonstrates that SocIoTy is practical for real-world smart homes. However, there are some issues that arise when attempting to deploy SocIoTy on commercial devices. Chief among them would be manufacturer support for the PartialEval functionality. Manufacturers would have to commit developer effort to program this functionality and integrate it into the devices, as well as ensure that SocIoTy's runtime does not interfere with the normal operation of the IoT device or artificially increase hardware requirements. The SocIoTy smartphone app would also require adequate domain separation to ensure the keys would not leak to other apps on the smartphone.

We believe that these issues are surmountable. Consumer interest in privacy is growing, and the additional guarantees of SocIoTy would be a marketable benefit for IoT products. Moreover, as our evaluation shows, SocIoTy induces limited overhead and resource costs, especially on more performant devices, which means that its addition will likely not require modifications to the device bill of materials. Smartphones also have numerous confidentiality mechanisms [73], which can be used to enforce domain separation for the keys. To reduce developer time, manufacturers can use our open-source implementation as a reference. Some IoT devices are even specifically designed for third-party functionality (*e.g.*, [12]), and we aim on porting SocIoTy to these platforms as future work. This will improve the ecosystem for SocIoTy and hopefully spur further exploration and adoption.

6 RELATED WORK

We now compare SocIoTy to other work with similar goals. We summarize our comparisons in Table 3.

Context-sensitive cryptography. In the literature, context-sensitive cryptography usually focuses on authentication, typically using some ambient information to establish a key or authorize an

Table 3: A comparison of related work with similar goals to those of SocIoTy.

Category	Authentication?	Encryption?	Binding?	Uses existing hardware?	Suitable for non-experts?
Context-sensitive auth [6, 14, 43, 60–62, 66]	✓	✗	✓	✗	✓
Proximity measurement [5, 72]	✓	✗	✗	✓	✗
Wearable devices [16, 21, 59]	✓	✗	✗	✓	✓
Geo-encryption [1, 25, 53, 57]	✗	✓	✓	✗	✗
Position-based crypto [13, 15, 17]	✓	✓	✓	N/A	✗
Time-specific encryption [34, 50, 52]	✗	✓	✓	N/A	✗
HSMs [31, 32, 69]	✓	✓	✓	✗	✗
SocIoTy (*this paper*)	✓	✓	✓	✓	✓

action. Early work by Mayrhofer and Gellersen [43] propose using mobile device acceleration information for authentication. Sigg [60] proposes an audio-based system as a case study of context-based security, and Sigg et al. [61] improve upon this scheme by applying fuzzy cryptography to handle noise in generated secrets. Wang et al. [66] use radio characteristics of BLE systems to set up context information for pairing purposes. Work in the HCI community [14] shows that users find context-sensitive schemes to be a promising improvement over traditional mechanisms.

An alternative to using the ambient features of a room is to use the presence of nearby systems as a context instead. Pico [62] is portable password storage hardware that pairs with other devices and applications to exchange keys and enable further seamless authentication. Such a device can be shaped as a watch, a key fob, a bracelet or an item of jewellery. Pico establishes a context when near dedicated Picosiblings devices, which coordinate and allow password actions on the primary Pico device. Instead of utilizing ambient features or dedicated hardware, SocIoTy re-uses existing IoT hardware (with a dedicated, unrelated purpose) to provide extra cryptographic functionalities.

Some context-sensitive authentication works focus on the IoT use case. Zhang et al. [72] describe an easier authentication for IoT devices by gesturing with a smartphone in close proximity to the devices. Aman et al. [5] used a similar concept for the authentication of IoT devices by accounting for physical location. These works do not provide a binding property for user data, however. Certain works employ IoT-specific characteristics for user authentication; in particular, [16, 21, 59] use wearable IoT devices as a second-factor for authentication. Anton et al. [6] propose context authentication for industrial IoT systems.

Location-sensitive cryptography. Heuristics around applying location information to cryptography were originally formed in the networking community, with a set of "geo-encryption" algorithms [1, 25, 53, 57] that introduce location and time as additional parameters to a cryptographic operation by using satellite data. More formal cryptographic definitions were introduced by Chandran et al. [17] as "position-based cryptography," wherein they demonstrate the impossibility of verifying the physical position (based on radio wave communications) of a number of colluding provers within a space in the standard model. Works since have explored the assumptions made by Chandran and their implications in complexity theory [13] and in the quantum setting [15].

Phuong et al. developed a location-based encryption scheme in 2019 [52]. However, their scheme requires bilinear maps (as used in an attribute-based encryption scheme) to achieve constant ciphertext size decryptable at arbitrary points within 2-D or 3-D grids. Further, they rely on time-specific encryption [34, 50] to ensure decryption only at particular points for a given ciphertext.

Hardware security modules (HSMs). Hardware security modules are separate, dedicated computing devices that protect cryptographic keys by storing them and monitoring their access and usage. They provide tamper-evidence or even tamper-resistance through the use of special hardware. Once tampering is detected, the device may stop functioning properly or delete its secret keys. HSMs can be used to protect keys used by certificate authorities, banks, and cryptocurrency wallets. They are present within vehicles [69], operational technology [32], and clouds [31]. HSMs act as trusted security anchors and gateway to the network. They securely generate, store, and process security-critical material shielded from any potentially malicious actor on the network and outside of it.

While providing good security guarantees on paper, historically HSMs have been too expensive for average consumers at the highest security levels and therefore have limited usability outside of large corporations [33, 35].

7 CONCLUSION

We present *SocIoTy*, an at-home cryptography system designed with non-technical users in mind. SocIoTy allows users to bind their secrets to their smart homes, giving them the opportunity to opt-in to additional protections for sensitive tasks. We protect against powerful, privacy-invading adversaries that can obtain the user's state or compromise their devices, all without requiring extra hardware. Our system protects against common surveillance techniques in this setting (like border searches), which is beyond what existing privacy-enhancing systems consider, all while providing the functionalities users expect, like authentication and encryption. Our benchmarks show that SocIoTy is practical, efficient, and conducive to deployment on real smart homes. In the future, we plan on exploring what other at-home services we can provide on top of IoT devices through systems like SocIoTy.

ACKNOWLEDGMENTS

The authors would like to acknowledge support from the NSF under awards 1653110, 1801479, 1955172 and 2030859, and from DARPA under contracts HR00112020021 and HR001120C0084. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors. Any mention of specific companies or products does not imply any endorsement by the authors, by their employers, or by the sponsors.

REFERENCES

[1] Ala Al-Fuqaha and Omar Al-Ibrahim. 2007. Geo-encryption protocol for mobile networks. *Computer Communications* 30, 11-12 (2007), 2510–2517.

[2] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. 2016. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In *ASIACRYPT 2016, Part I (LNCS, Vol. 10031)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.). Springer, Heidelberg, 191–219. https://doi.org/10.1007/978-3-662-53887-6_7

[3] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *EUROCRYPT 2015, Part I (LNCS, Vol. 9056)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Heidelberg, 430–454. https://doi.org/10.1007/978-3-662-46800-5_17

[4] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. 2020. Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols. *IACR Trans. Symm. Cryptol.* 2020, 3 (2020), 1–45. https://doi.org/10.13154/tosc.v2020.i3.1-45

[5] Muhammad Naveed Aman, Mohamed Haroon Basheer, and Biplab Sikdar. 2018. Two-factor authentication for IoT with location information. *IEEE Internet of Things Journal* 6, 2 (2018), 3335–3351.

[6] Simon Duque Anton, Daniel Fraunholz, Christoph Lipps, Khurshid Alam, and Hans Dieter Schotten. 2019. Putting things in context: Securing industrial authentication with context information. *arXiv preprint arXiv:1905.12239* (2019).

[7] Apple Inc. 2021. Apple TV HD Technical Specifications. https://support.apple.com/kb/SP724. Accessed 2/27/2023.

[8] Tomer Ashur and Siemen Dhooghe. 2018. MARVELlous: a STARK-Friendly Family of Cryptographic Primitives. Cryptology ePrint Archive, Report 2018/1098. https://eprint.iacr.org/2018/1098.

[9] Bastian Jaansen. 2020. Authenticator. https://github.com/BastiaanJansen/Authenticator.

[10] M. Bellare and R. Impagliazzo. 1999. A tool for obtaining tighter security analyses of pseudorandom function based constructions, with applications to PRP to PRF conversion. Cryptology ePrint Archive, Report 1999/024. https://eprint.iacr.org/1999/024.

[11] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *20th ACM STOC*. ACM Press, 1–10. https://doi.org/10.1145/62212.62213

[12] B&H Photo. 2023. Amazon Echo Dot (3rd Generation, Charcoal). https://www.bhphotovideo.com/c/product/1437065-REG/amazon_b0792kthkj_echo_dot_3rd_generation.html/specs. Accessed 2/27/2023.

[13] Joshua Brody, Stefan Dziembowski, Sebastian Faust, and Krzysztof Pietrzak. 2017. Position-Based Cryptography and Multiparty Communication Complexity. In *TCC 2017, Part I (LNCS, Vol. 10677)*, Yael Kalai and Leonid Reyzin (Eds.). Springer, Heidelberg, 56–81. https://doi.org/10.1007/978-3-319-70500-2_3

[14] Matthias Budde, Till Riedel, Marcel Köpke, Matthias Berning, and Michael Beigl. 2014. A Comparative Study to Evaluate the Usability of Context-based Wi-Fi Access Mechanisms. In *Universal Access in Human-Computer Interaction. Aging and Assistive Environments: 8th International Conference, UAHCI 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014, Proceedings, Part III 8*. Springer, 451–462.

[15] Harry Buhrman, Nishanth Chandran, Serge Fehr, Ran Gelles, Vipul Goyal, Rafail Ostrovsky, and Christian Schaffner. 2011. Position-Based Quantum Cryptography: Impossibility and Constructions. In *CRYPTO 2011 (LNCS, Vol. 6841)*, Phillip Rogaway (Ed.). Springer, Heidelberg, 429–446. https://doi.org/10.1007/978-3-642-22792-9_24

[16] Yetong Cao, Qian Zhang, Fan Li, Song Yang, and Yu Wang. 2020. PPGPass: Nonintrusive and secure mobile two-factor authentication via wearables. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1917–1926.

[17] Nishanth Chandran, Vipul Goyal, Ryan Moriarty, and Rafail Ostrovsky. 2009. Position Based Cryptography. In *CRYPTO 2009 (LNCS, Vol. 5677)*, Shai Halevi (Ed.). Springer, Heidelberg, 391–407. https://doi.org/10.1007/978-3-642-03356-8_23

[18] David Chaum, Claude Crépeau, and Ivan Damgård. 1988. Multiparty Unconditionally Secure Protocols (Abstract) (Informal Contribution). In *CRYPTO'87 (LNCS, Vol. 293)*, Carl Pomerance (Ed.). Springer, Heidelberg, 462. https://doi.org/10.1007/3-540-48184-2_43

[19] Chrysta Cherrie. 2021. The 2021 State of the Auth Report: 2FA Climbs, While Password Managers and Biometrics Trend. https://duo.com/blog/the-2021-state-of-the-auth-report-2fa-climbs-password-managers-biometrics-trend. Accessed 2022-07-29.

[20] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. 2021. Fluid MPC: Secure Multiparty Computation with Dynamic Participants. In *CRYPTO 2021, Part II (LNCS, Vol. 12826)*, Tal Malkin and Chris Peikert (Eds.). Springer, Heidelberg, Virtual Event, 94–123. https://doi.org/10.1007/978-3-030-84245-1_4

[21] John Chuang. 2014. One-step two-factor authentication with wearable biosensors. In *Symposium on Usable Privacy and Security-SOUPS*, Vol. 14.

[22] Cisco. 2020. Configure AnyConnect Secure Mobility Client using One-Time Password (OTP) for Two-Factor Authentication on an ASA. https://www.cisco.com/c/en/us/support/docs/security/anyconnect-secure-mobility-client/213931-configure-anyconnect-secure-mobility-cli.html. Accessed 2022-07-29.

[23] Jyoti Deogirikar and Amarsinh Vidhate. 2017. Security attacks in IoT: A survey. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*. IEEE, 32–37.

[24] Duo. 2021. Duo Authentication for Epic. https://duo.com/docs/epic. Accessed 7/29/2022.

[25] Mahdi Daghmechi Firoozjaei and Javad Vahidi. 2012. Implementing geo-encryption in GSM cellular network. In *2012 9th International Conference on Communications (COMM)*. IEEE, 299–302.

[26] Shoni Gilboa, Shay Gueron, and Ben Morris. 2018. How many queries are needed to distinguish a truncated random permutation from a random function? *Journal of Cryptology* 31, 1 (2018), 162–171.

[27] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1984. How to Construct Random Functions (Extended Abstract). In *25th FOCS*. IEEE Computer Society Press, 464–479. https://doi.org/10.1109/SFCS.1984.715949

[28] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *19th ACM STOC*, Alfred Aho (Ed.). ACM Press, 218–229. https://doi.org/10.1145/28395.28420

[29] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *USENIX Security 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 519–535.

[30] Chris Hall, David Wagner, John Kelsey, and Bruce Schneier. 1998. Building PRFs from PRPs. In *CRYPTO'98 (LNCS, Vol. 1462)*, Hugo Krawczyk (Ed.). Springer, Heidelberg, 370–389. https://doi.org/10.1007/BFb0055742

[31] Juhyeng Han, Seongmin Kim, Taesoo Kim, and Dongsu Han. 2019. Toward scaling hardware security module for emerging cloud services. In *Proceedings of the 4th Workshop on System Software for Trusted Execution*. 1–6.

[32] William Hupp, Adarsh Hasandka, Ricardo Siqueira de Carvalho, and Danish Saleem. 2020. Module-OT: a hardware security module for operational technology. In *2020 IEEE Texas Power and Energy Conference (TPEC)*. IEEE, 1–6.

[33] SANS institution. 2002. Global Information Assurance Certification Paper. https://www.giac.org/paper/gsec/1508/overview-hardware-security-modules/102811.

[34] Kohei Kasamatsu, Takahiro Matsuda, Keita Emura, Nuttapong Attrapadung, Goichiro Hanaoka, and Hideki Imai. 2012. Time-specific encryption from forward-secure encryption. In *International Conference on Security and Cryptography for Networks*. Springer, 184–204.

[35] Anand Kashyap. 2018. The Next Generation: HSM approach delivers unparalleld cost/benefit for organizations. https://securitytoday.com/Articles/2018/12/01/The-Next-Generation.aspx?Page=1.

[36] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 1575–1590. https://doi.org/10.1145/3372297.3417872

[37] Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl, Eduardo Soria-Vazquez, and Srinivas Vivek. 2017. Faster Secure Multi-party Computation of AES and DES Using Lookup Tables. In *ACNS 17 (LNCS, Vol. 10355)*, Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi (Eds.). Springer, Heidelberg, 229–249. https://doi.org/10.1007/978-3-319-61204-1_12

[38] Hugo Krawczyk. 2010. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In *CRYPTO 2010 (LNCS, Vol. 6223)*, Tal Rabin (Ed.). Springer, Heidelberg, 631–648. https://doi.org/10.1007/978-3-642-14623-7_34

[39] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. 1997. HMAC: Keyed-Hashing for Message Authentication. IETF Internet Request for Comments 2104.

[40] Sam Kumar, Yuncong Hu, Michael P. Andersen, Raluca Ada Popa, and David E. Culler. 2019. JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT. In *USENIX Security 2019*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 1519–1536.

[41] Dawn MacKeen. 2022. Can New Technology Make Home Dialysis a More Realistic Option? *The New York Times* (10 Nov 2022). Accessed 2023-02-27.

[42] Matt Burns. 2011. Nest Thermostat Teardown Reveals Beautiful Innards, Powerful ARM CPU, Zigbee Radio. https://techcrunch.com/2011/12/22/nest-arm-zigbee/. Accessed 2/27/2023.

[43] Rene Mayrhofer and Hans Gellersen. 2007. Shake well before use: two implementations for implicit context authentication. *Adjunct Proc. Ubicomp 2007* (2007).

[44] Mordor Intelligence. 2022. Global smart homes market—growth, analysis, forecast to 2022. https://www.mordorintelligence.com/industry-reports/global-smart-homes-market-industry.

[45] David M'Raihi, Mihir Bellare, Frank Hoornaert, David Naccache, and Ohad Ranen. 2005. RFC 4226: HOTP: An hmac-based one-time password algorithm.

[46] David M'Raihi, Salah Machani, Mingliang Pei, and Johan Rydell. 2011. RFC 6238: TOTP: Time-based one-time password algorithm.

[47] Moni Naor, Benny Pinkas, and Omer Reingold. 1999. Distributed Pseudo-random Functions and KDCs. In *EUROCRYPT'99 (LNCS, Vol. 1592)*, Jacques Stern (Ed.).

Springer, Heidelberg, 327–346. https://doi.org/10.1007/3-540-48910-X_23

[48] NIST. 2023. Lightweight Cryptography Standardization Process: NIST Selects Ascon. https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon. Accessed 2/27/2023.

[49] Kim Parker, Juliana Menasce Horowitz, and Rachel Minkin. 2020. How the Coronavirus Outbreak Has - and Hasn't - Changed the Way Americans Work. *Pew Research Center* (9 Dec 2020). Accessed 2023-02-27.

[50] Kenneth G Paterson and Elizabeth A Quaglia. 2010. Time-specific encryption. In *International Conference on Security and Cryptography for Networks*. Springer, 1–16.

[51] Pew Research Center. 2021. Mobile Fact Sheet. https://www.pewresearch.org/internet/fact-sheet/mobile/. Accessed 7/29/2022.

[52] Tran Viet Xuan Phuong, Willy Susilo, Guomin Yang, Jun Yan, and Dongxi Liu. 2019. Location Based Encryption. In *ACISP 19 (LNCS, Vol. 11547)*, Julian Jang-Jaccard and Fuchun Guo (Eds.). Springer, Heidelberg, 21–38. https://doi.org/10.1007/978-3-030-21548-4_2

[53] Di Qiu, Sherman Lo, Per Enge, Dan Boneh, and Ben Peterson. 2007. Geoencryption using loran. In *Proceedings of the 2007 National Technical Meeting of The Institute of Navigation*. 104–115.

[54] Robinhood. 2022. Two-Factor Authentication. https://robinhood.com/us/en/support/articles/twofactor-authentication/. Accessed 2022-07-29.

[55] Bryan Robinson. 2022. Remote Work is Here to Stay And Will Increase Into 2023, Experts Say. *Forbes* (01 Feb 2022).

[56] Sarah Scheffler and Mayank Varia. 2021. Protecting Cryptography Against Compelled Self-Incrimination. In *USENIX Security 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 591–608.

[57] Logan Scott and Dorothy E Denning. 2003. A location based encryption technique and some of its applications. In *Proceedings of the 2003 National Technical Meeting of The Institute of Navigation*. 734–740.

[58] Jayasree Sengupta, Sushmita Ruj, and Sipra Das Bit. 2020. A comprehensive survey on attacks, security issues and blockchain solutions for IoT and IIoT. *Journal of Network and Computer Applications* 149 (2020), 102481.

[59] Prakash Shrestha and Nitesh Saxena. 2018. Listening watch: Wearable two-factor authentication using speech signals resilient to near-far attacks. In *Proceedings of the 11th ACM conference on security & privacy in wireless and mobile networks*. 99–110.

[60] Stephan Sigg. 2011. Context-based security: State of the art, open research topics and a case study. In *Proceedings of the 5th ACM International Workshop on Context-Awareness for Self-Managing Systems*. 17–23.

[61] Stephan Sigg, Dominik Schuermann, and Yusheng Ji. 2012. Pintext: A framework for secure communication based on context. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services: 8th International ICST Conference, MobiQuitous 2011, Copenhagen, Denmark, December 6-9, 2011, Revised Selected Papers 8*. Springer, 314–325.

[62] Frank Stajano. 2011. Pico: No more passwords!. In *Security Protocols XIX: 19th International Workshop, Cambridge, UK, March 28-30, 2011, Revised Selected Papers 19*. Springer, 49–81.

[63] Statista. 2022. Digital market - Smart Home. https://www.statista.com/outlook/dmo/smart-home/worldwide.

[64] TechInfoDepot. 2023. Belkin WeMo Light Switch (F7C030). http://en.techinfodepot.shoutwiki.com/wiki/Belkin_WeMo_Light_Switch_(F7C030). Accessed 2/27/2023.

[65] Nirvan Tyagi, Muhammad Haris Mughees, Thomas Ristenpart, and Ian Miers. 2018. BurnBox: Self-Revocable Encryption in a World Of Compelled Access. In *USENIX Security 2018*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 445–461.

[66] Juan Wang, Karim Lounis, and Mohammad Zulkernine. 2019. CSKES: a context-based secure keyless entry system. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. IEEE, 817–822.

[67] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Global-Scale Secure Multiparty Computation. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 39–56. https://doi.org/10.1145/3133956.3133979

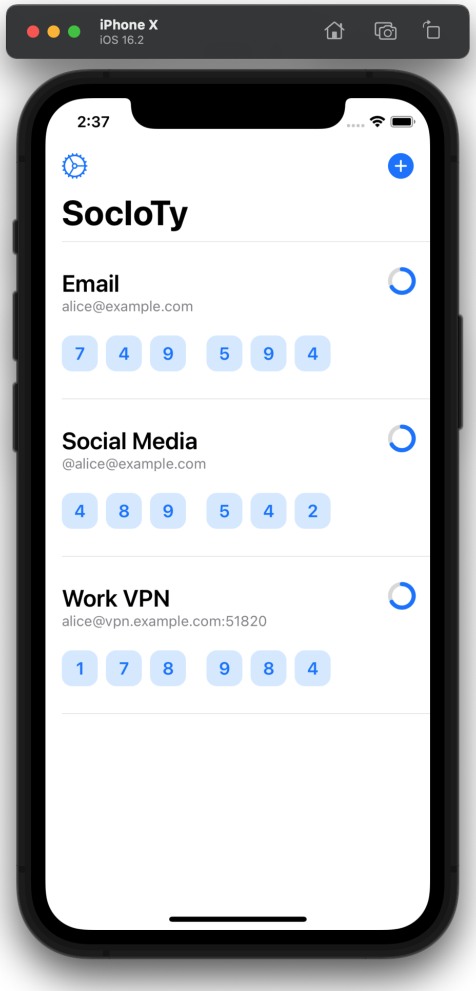
[68] Yash Wate. 2021. How to Enable Two-Factor Authentication on Facebook, Instagram, and Twitter. https://techpp.com/2020/02/03/enable-two-factor-authentication-instagram-facebook-twitter/. Accessed 2022-07-29.

[69] Marko Wolf and Timo Gendrullis. 2011. Design, implementation, and evaluation of a vehicular hardware security module. In *International Conference on Information Security and Cryptology*. Springer, 302–318.

[70] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *27th FOCS*. IEEE Computer Society Press, 162–167. https://doi.org/10.1109/SFCS.1986.25

[71] C. Bormann Z. Shelby, K. Hartke. 2014. *The Constrained Application Protocol (CoAP)*. RFC 7252. RFC Editor. https://www.rfc-editor.org/rfc/rfc7252

[72] Jiansong Zhang, Zeyu Wang, Zhice Yang, and Qian Zhang. 2017. Proximity based IoT device authentication. In *IEEE INFOCOM 2017-IEEE conference on computer communications*. IEEE, 1–9.



SocIoTy

Email
alice@example.com
7 4 9 5 9 4

Social Media
@alice@example.com
4 8 9 5 4 2

Work VPN
alice@vpn.example.com:51820
1 7 8 9 8 4

Figure 10: A Simulator screenshot of our iOS app. Note that all benchmarks were performed with a hardware iPhone X.

[73] Maximilian Zinkus, Tushar M. Jois, and Matthew Green. 2022. SoK: Cryptographic Confidentiality of Data on Mobile Devices. *PoPETs* 2022, 1 (Jan. 2022), 586–607. https://doi.org/10.2478/popets-2022-0029

A ADDITIONAL EVALUATION RESULTS

We present additional results from our evaluation in this section.

ESP32 Gen and Recon. As discussed in Section 5.1, we believe that it is unlikely for Gen and Recon to be run on devices that have computational resources below that of the Pi Zero. However, for completeness we evaluated how Gen and Recon fare on the ESP32 for different configurations of (n, t). These results are shown in Tables 4 and 5, respectively. Note that $\bar{x}$ is the average and s is the standard deviation of each execution.

Scalability with CoAP. CoAP scalability results for more configurations of (n, t) are shown in Table 6. Note that there is a sharp

Table 4: Microbenchmarks for Gen on the ESP32 devices over varying configurations of total number of parties n and reconstruction threshold t. All times are in milliseconds.

Experiment	$\bar{x}$	s
Gen ($n = 5, t = 3$)	41.16	0.02
Gen ($n = 5, t = 4$)	41.63	0.02
Gen ($n = 5, t = 5$)	42.12	0.02
Gen ($n = 6, t = 4$)	49.78	0.02
Gen ($n = 6, t = 5$)	50.33	0.02
Gen ($n = 6, t = 6$)	50.89	0.02
Gen ($n = 7, t = 5$)	58.55	0.02
Gen ($n = 7, t = 6$)	59.18	0.02
Gen ($n = 7, t = 7$)	59.83	0.02
Gen ($n = 8, t = 6$)	67.48	0.02
Gen ($n = 8, t = 7$)	68.20	0.02
Gen ($n = 8, t = 8$)	68.92	0.02
Gen ($n = 9, t = 7$)	76.58	0.02
Gen ($n = 9, t = 8$)	77.36	0.02
Gen ($n = 9, t = 9$)	78.17	0.02
Gen ($n = 10, t = 8$)	85.83	0.02
Gen ($n = 10, t = 9$)	86.70	0.02
Gen ($n = 10, t = 10$)	87.73	0.02
Gen ($n = 11, t = 9$)	95.25	0.02
Gen ($n = 11, t = 10$)	96.18	0.02
Gen ($n = 11, t = 11$)	97.31	0.02
Gen ($n = 12, t = 10$)	104.81	0.02
Gen ($n = 12, t = 11$)	105.83	0.02
Gen ($n = 12, t = 12$)	107.05	0.02

Table 5: Microbenchmarks for Recon on the ESP32 devices over varying configurations of total number of parties n and reconstruction threshold t. All times are in milliseconds.

Experiment	$\bar{x}$	s
Recon ($n = 5, t = 3$)	382.47	0.01
Recon ($n = 5, t = 4$)	382.47	0.01
Recon ($n = 5, t = 5$)	382.11	0.01
Recon ($n = 6, t = 4$)	495.02	0.01
Recon ($n = 6, t = 5$)	495.02	0.01
Recon ($n = 6, t = 6$)	494.48	0.01
Recon ($n = 7, t = 5$)	622.77	0.01
Recon ($n = 7, t = 6$)	622.77	0.01
Recon ($n = 7, t = 7$)	622.02	0.01
Recon ($n = 8, t = 6$)	765.77	0.01
Recon ($n = 8, t = 7$)	765.77	0.01
Recon ($n = 8, t = 8$)	764.76	0.01
Recon ($n = 9, t = 7$)	923.98	0.01
Recon ($n = 9, t = 8$)	923.98	0.01
Recon ($n = 9, t = 9$)	922.68	0.01
Recon ($n = 10, t = 8$)	1097.46	0.01
Recon ($n = 10, t = 9$)	1097.46	0.01
Recon ($n = 10, t = 10$)	1097.44	0.01
Recon ($n = 11, t = 9$)	1286.10	0.01
Recon ($n = 11, t = 10$)	1286.51	0.01
Recon ($n = 11, t = 11$)	1286.09	0.01
Recon ($n = 12, t = 10$)	1490.53	0.01
Recon ($n = 12, t = 11$)	1490.53	0.01
Recon ($n = 12, t = 12$)	1490.06	0.01

increase when the number of nodes is $n \geq 10$ and the threshold is $t = n$. In this case, we found that one of the Raspberry Pi Zeros performs significantly worse on network communication than all other devices, perhaps due to a manufacturing issue. As mentioned in Section 5.3, when the threshold is sufficiently large to encompass the slowest of devices, the computation becomes bounded by the slowest performing devices.

iOS App. A screenshot of the iOS app we built for our end-to-end deployment evaluation in Section 5.3 can be found in Figure 10.

Table 6: Protocol execution time using CoAP over all evaluated configurations of total number of parties n and reconstruction threshold t. All times are in milliseconds.

Experiment	$\bar{x}$	s
($n = 5, t = 3$)	7.90	1.23
($n = 5, t = 4$)	8.96	1.76
($n = 5, t = 5$)	10.77	4.34
($n = 6, t = 4$)	8.85	1.76
($n = 6, t = 5$)	10.65	2.26
($n = 6, t = 6$)	12.05	2.16
($n = 7, t = 5$)	11.05	2.51
($n = 7, t = 6$)	12.57	2.65
($n = 7, t = 7$)	14.41	3.21
($n = 8, t = 6$)	12.39	2.38
($n = 8, t = 7$)	14.31	2.74
($n = 8, t = 8$)	16.06	2.99
($n = 9, t = 7$)	14.42	2.51
($n = 9, t = 8$)	16.70	3.47
($n = 9, t = 9$)	18.37	4.03
($n = 10, t = 8$)	17.07	4.28
($n = 10, t = 9$)	19.74	4.21
($n = 10, t = 10$)	49.57	151.41
($n = 11, t = 9$)	19.81	3.90
($n = 11, t = 10$)	22.35	3.97
($n = 11, t = 11$)	48.05	134.40
($n = 12, t = 10$)	23.34	4.87
($n = 12, t = 11$)	24.63	5.01
($n = 12, t = 12$)	53.62	145.06

B MPC-BASED SOCIOTY

In this appendix, we explore how SocIoTy could be built using secure multi-party computation (MPC) and investigate its execution time, comparing it to our dual-layered PRF solution presented in the main body of this work.

MPC [11, 18, 28, 70] allows a set of parties to compute a function of their inputs while keeping those inputs confidential. Generic MPC protocols Π can securely compute arbitrary functions f, described as either Boolean or arithmetic circuits. Of course, the security and properties provided by Π may differ widely based on the protocol chosen.

MPC for IoT devices. In the main body of this work we build SocIoTy from a combination of a TDPRF (on the smart home devices) and a regular PRF (on the smartphone) to mitigate the impact of device compromise. To do the same with MPC, we evaluate the standard cryptographic functions used in authentication and encryption within an MPC protocol. Specifically, the secret key material is secret shared among all n IoT devices, and only ever reconstructed *within* the MPC. Thus, like in the proposed SocIoTy model, compromise of one device—or even a handful of devices—does not destroy the security guarantees of the system.

We utilize an MPC system that allows for *dynamic participation*—entering and exiting—as a part of protocol execution. MPC computations are potentially long-running and involve several steps of interaction between parties, and it may be unrealistic to expect parties (especially low-powered ones) to participate for the duration of the whole protocol. So, with dynamic participation, a party (*i.e.*, an IoT device) that must leave (*e.g.*, to service a user event), can do so without seriously impacting protocol execution. We choose the recent proposal of Choudhuri et al. [20], *Fluid MPC*, in which the execution of an MPC protocol is divided into discrete

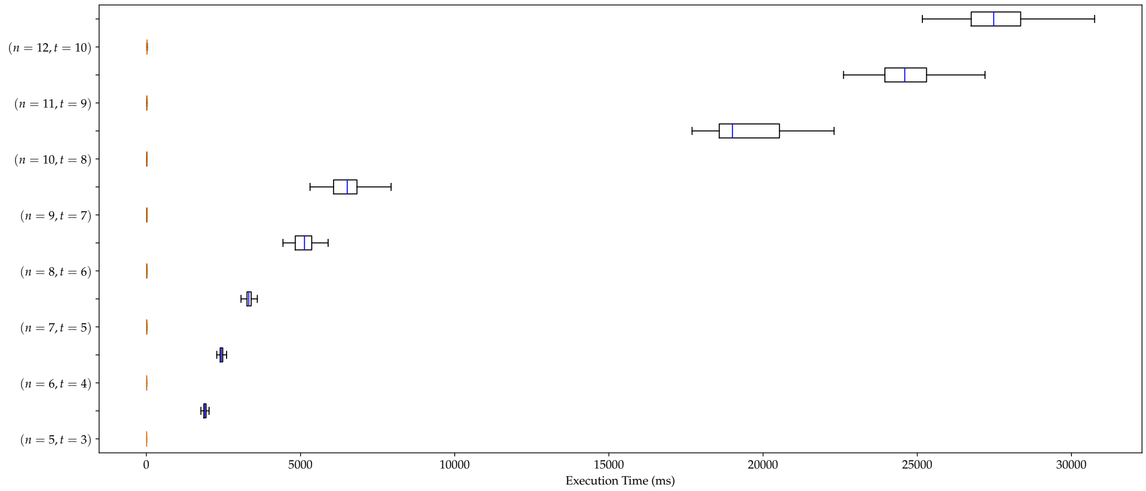


Figure 11: Comparison of execution time boxplots for the TDPRF-based SocIoTy protocol (lower boxplot, orange median) and the MPC-based SocIoTy protocol (upper boxplot, blue median).

stages known as "epochs" such that a different set of parties, called a "committee," participates in each epoch. The Fluid MPC protocol—a variant of BGW [11]—achieves *maximal fluidity*, *i.e.*, each epoch has one round of communication.

MPC-friendly PRFs. The SocIoTy solution presented in the main body of this work builds its dual-layered PRF from a TDPRF, which in turn is build with elliptic curve multiplications (see Section 4.5). However, these kinds of Diffie-Hellman operations are too slow to operate inside of an MPC circuit, and we require different primitives for the distributed computation. So, in this appendix, we use the symmetric cipher MiMC [2] to build a PRF for TOTP inside of MPC. We study MiMC because it is a relatively new, MPC-friendly cipher with low multiplicative complexity; similar ciphers have been explored in [3, 4, 8, 29]. For MiMC, we represent our computation as an arithmetic circuit over $GF(2^{128})$.

Block ciphers are typically modeled as a pseudorandom *permutation* (PRP), but there is a line of work in the theoretical literature [10, 26, 30] on how truncating the output of a PRP results in a (secure) PRF. As defined [46], TOTP truncates the output of its PRF to be 6 digits long. Thus, as long as our PRP-to-PRF truncation results in enough bits to be further truncated to be 6 digits long (PRF mod 10^6), we can use block ciphers as the PRF in TOTP. With this in mind, we set $\mathsf{PRF.Eval}(sk, t) = \mathsf{Enc}(sk, t)[: 24]$, where Enc is the symmetric key encryption function of a block cipher like MiMC and the output is truncated to be 24 bits long. Since $2^{24} > 10^6$, truncating the output to 24 bits will still allow for OTPs that are at least 6 decimal digits, meeting the current interface for a TOTP.

Evaluation. With all of this in mind, we can evaluate how an MPC-based SocIoTy construction would fare in the real world. Our implementation consists of a BGW MPC [11] with extensions to support churn derived from Fluid MPC [20]. The underlying MPC requires communication between nodes to compute the result of a multiplication (referred to as π_{mult} in [20]), as well as to hand-off shares to another party if exiting the network (π_{trans} in [20]). Communication between device is handled via a gRPC protocol that transmits share data between parties. We wrote a circuit $\mathsf{TOTP}^{\mathsf{MiMC}}_{\mathrm{home}}$ that implements the TOTP evaluation using MiMC to build the PRF.

Table 7: MPC-based SocIoTy time distributions for varying the (party size, threshold) pairs. All times are in milliseconds.

Experiment	$\mathsf{TOTP}^{\mathsf{MiMC}}_{\mathrm{home}}$ (MPC)
$(n = 5, t = 3)$	$\bar{x} = 1903.05$, $s = 60.63$
$(n = 6, t = 4)$	$\bar{x} = 2464.53$, $s = 170.26$
$(n = 7, t = 5)$	$\bar{x} = 3342.50$, $s = 116.07$
$(n = 8, t = 6)$	$\bar{x} = 5103.57$, $s = 317.38$
$(n = 9, t = 7)$	$\bar{x} = 6461.75$, $s = 508.85$
$(n = 10, t = 8)$	$\bar{x} = 19435.92$, $s = 1129.71$
$(n = 11, t = 9)$	$\bar{x} = 24728.66$, $s = 1031.36$
$(n = 12, t = 10)$	$\bar{x} = 28632.24$, $s = 10212.52$

We present a benchmark of the MiMC protocol execution on a smaller version of our simulated smart home used in Section 5, with 6 Raspberry Pi 3B+ devices, 3 Raspberry Pi 2B devices, and 3 Raspberry Pi Zero devices, for a total of 12 devices. To align our comparison to the evaluation presented in Section 5, we use the same configurations of (n, t) as the benchmarks in Figure 8.

The results of this evaluation can be found in Table 7, which lists results for the average $\bar{x}$ and standard deviation s of the MPC protocol executions at each configuration. Note that executions take multiple seconds even at small (party size, threshold) pairs. While our proposed SocIoTy construction was efficient enough for end-to-end deployment on even an ESP32 (Section 5.3), we were unable to effectively run the MPC implementation on smaller classes of devices than the Raspberry Pi 3B+ and 2B. Also, our proposed construction scaled well to even 17 participants in the network, while the MPC approach slows considerably even at $n = 8$, likely due to its quadratic computation complexity. Lastly starting at $n = 10$, the Raspberry Pi Zero nodes join the protocol, slowing the computation significantly.

Our results are visualized in Figure 11. For each (n, t) configuration, we plotted both the TDPRF-based SocIoTy (from Figure 8) and our MPC-based SocIoTy execution results. Within the results for a configuration, the lower boxplot with an orange median is the TDPRF result, and the upper boxplot with a blue median is the MPC result. Clearly, the MPC-based SocIoTy evaluation is has a much higher execution time (by several orders of magnitude) – and is more variable – when compared to our proposed TDPRF approach. Thus, based on this experimental analysis, and despite several MPC-specific optimizations, an MPC-based SocIoTy solution is clearly far too inefficient for our use case.