Ad-Hoc Secure Two-Party Computation on Mobile Devices using Hardware Tokens

(Full Version)*

Daniel Demmler, Thomas Schneider, and Michael Zohner

Technische Universität Darmstadt, Germany
{daniel.demmler,thomas.schneider,michael.zohner}@ec-spride.de

July 3, 2014

Abstract

Secure two-party computation allows two mutually distrusting parties to jointly compute an arbitrary function on their private inputs without revealing anything but the result. An interesting target for deploying secure computation protocols are mobile devices as they contain a lot of sensitive user data. However, their resource restriction makes the deployment of secure computation protocols a challenging task.

In this work, we optimize and implement the secure computation protocol by Goldreich-Micali-Wigderson (GMW) on mobile phones. To increase performance, we extend the protocol by a trusted hardware token (i.e., a smartcard). The trusted hardware token allows to pre-compute most of the workload in an initialization phase, which is executed locally on one device and can be pre-computed independently of the later communication partner. We develop and analyze a proof-of-concept implementation of generic secure two-party computation on Android smart phones making use of a microSD smartcard. Our use cases include private set intersection for finding shared contacts and private scheduling of a meeting with location preferences. For private set intersection, our token-aided implementation on mobile phones is up to two orders of magnitude faster than previous generic secure two-party computation protocols on mobile phones and even as fast as previous work on desktop computers.

1 Introduction

Secure two-party computation allows two parties to process their sensitive data in such a way that its privacy is protected. In the late eighties, Yao's garbled circuits protocol [Yao86] and the protocol of Goldreich-Micali-Wigderson (GMW) [GMW87] showed the feasibility of secure computation. However, secure computation was considered to be mostly of theoretical interest until the Fairplay framework [MNPS04] demonstrated that it is indeed practical. Since then, many optimizations have been proposed and several frameworks have implemented Yao's garbled circuits protocol (e.g., FastGC [HEKM11]) and the GMW protocol (e.g., the framework of [CHK+12]) on desktop PCs.

Motivated by the advances of secure computation on desktop PCs, researchers have started to investigate whether secure computation can also be performed in the mobile domain. Mobile devices, in particular smartphones, are an excellent environment for secure computation, since they accompany users in their daily lives and typically hold contact information, calendars, and photos. Users also store sensitive data, such as passwords or banking information on their devices. Moreover, typical smartphones are equipped with a multitude of sensors that collect a lot of sensitive information about their users' contexts. Therefore, it is of special importance to protect the privacy of data handled in the mobile domain.

In contrast to desktop PCs, mobile devices are rather limited in computational power, available memory, communication capabilities, and most notably battery life. Although mobile phones have seen an increase in processing speed over the past years, they are still about one order of magnitude slower than typical desktop computers when evaluating cryptographic primitives (cf. §5.4). These differences are due to the CPU architectures having a more restrictive instruction set and being optimized for low power consumption rather than performance, since mobile devices are battery-powered and lack active cooling.

*Please cite the conference version of this work published at USENIX Security'14 [DSZ14].

Moreover, the limited size of the main memory requires the programmer to carefully handle data objects in order to avoid costly garbage collections on Java-based Android smartphones. Network connections of mobile devices are almost exclusively established via wireless connections that have lower bandwidth and higher, often varying latency compared to wired connections. Tasks that are computationally intensive or require long send/receive operations should be avoided when a mobile device is running on battery, as such tasks quickly drain the battery charge and thereby reduce the phone's standby time. Instead, such operations could be pre-computed when the mobile device is connected to a power source, which usually happens overnight. These limitations pose a big challenge for efficient secure computation and cause generic secure computation protocols to be several hundred times slower on mobile devices than on desktop PCs [HCE11], even in the semi-honest adversary model.

To enable secure two-party computation in the mobile domain, solutions have been developed that outsource secure computation to the cloud, e.g., [KMR12, Hua12, CMTB13]. However, recent events have shown that cloud service providers can be forced to give away data to third parties that are not necessarily trusted, such as foreign government agencies. Even if the employed protocols ensure that the cloud provider learns no information about the users' sensitive data, he can still learn and hence be forced to reveal meta-information such as the frequency of access, communication partners involved, the computed function, or the size of the inputs. Moreover, these server-aided approaches require the mobile device to be connected to the Internet which might not be possible in every situation or may cause additional costs.

An alternative solution, which we also use in this work, is to outsource expensive operations to a trusted hardware token that has very limited computational resources and is locally held by one of the communication partners.[1] Such hardware tokens are increasingly being adopted in practice, e.g., trusted platform modules (TPMs). Their adoption is particularly noteworthy on mobile devices in the form of smartcards that are the basis for subscriber identity modules (SIM cards), as well as for mobile payment or ticketing systems. A first approach for outsourcing Yao's garbled circuits protocol to such a trusted hardware token was proposed in [JKSS10]. However, this protocol requires the function to be known in advance and uses costly symmetric cryptographic operations during the online phase. We give an alternative solution that removes these drawbacks.

[1] This locality is also a security feature, as external adversaries either need to corrupt the token before it is shipped to the user or later get physical access to break into it.

1.1 Outline and Our Contributions

In this work, we introduce a scheme for token-aided ad-hoc generic secure two-party computation on mobile devices based on the GMW protocol. After introducing preliminaries (§2) we detail our setting and trust assumptions that are similar to the ones in a TPM scenario (§3). We outline how a trusted hardware token can be used to shift major parts of the workload into an initialization phase that can be pre-computed on the token, independently of the later communication partner (§4), e.g., while the mobile device is charging. We thereby obtain a token-aided scheme that is well-suited for efficient and decentralized (ad-hoc) secure computation in the mobile domain. We implement and evaluate our scheme (§5) and demonstrate its performance using typical secure computation applications for mobile devices, such as securely scheduling a meeting with location preferences and privacy-preserving set intersection (§6). We compare our scheme to related work (§7) and conclude and present directions for future work (§8). More detailed, our contributions are as follows.

Token-Aided Ad-Hoc Secure Two-Party Computation on Mobile Devices (§4) We develop a token-aided secure computation protocol which offloads the main workload of the GMW protocol to a pre-computation phase by introducing a secure hardware token $\mathcal{T}$, held by one party $\mathcal{A}$ (cf. §3). $\mathcal{T}$ is issued by a trusted third party and provides correlated randomness [Hua12, Chap. 6] to both parties that is later used in the secure computation protocol. To prepare the secure computation, the other party $\mathcal{B}$ obtains seeds for his part of the correlated randomness from $\mathcal{T}$ via an encrypted channel. To further increase flexibility, we describe how to make the pre-computation independent of the size of the evaluated function $|f|$, at the cost of a $t \cdot \log_2 |f|$ factor communication overhead between $\mathcal{T}$ and $\mathcal{B}$, where t is the symmetric security parameter. In contrast to Yao-based approaches [MNPS04, JKSS10, HCE11, HEK12] and previous realizations of the GMW protocol [CHK+12, SZ13, ALSZ13], our protocol offers several benefits as summarized in Tab. 1 (cf. §4.5 for details).

Table 1: Comparison with related work.

Property	**Yao** [HCE11]	**Token Yao** [JKSS10]	**GMW** [CHK+12]	**Ours** §4
f unknown in init phase	✓	✗	✓	✓
ad-hoc communication $\ll t \cdot \lvert f \rvert$	✗	✓	✗	✓
$\ll \lvert f \rvert$ crypto operations in ad-hoc phase	✗	✗	✗	✓

Implementation (§5) We implement our token-aided protocol for semi-honest participants and evaluate its performance using two consumer-grade Android smartphones and an off-the-shelf smartcard. Thereby, we provide an estimate for the achievable runtime of generic secure computation in the mobile domain. Our implementation enables a developer to specify the functionality as a Boolean circuit, which can, for instance, be generated from a high-level specification language. We show that the performance of our token-aided pre-computation phase is comparable to interactively generating the correlated randomness using oblivious transfer.

Applications (§6) We demonstrate the practical feasibility of the GMW protocol on mobile devices by performing secure two-party computation on two smartphones using various privacy-preserving applications such as availability scheduling (§6.1), location-aware scheduling (§6.2), and set-intersection (§6.3). Most notably, for private set-intersection, our token-aided scheme outperforms related work that evaluates generic secure computation schemes on mobile devices [HCE11] by up to two orders of magnitude and has a performance that is comparable with secure computation schemes that are executed in a desktop environment [HEK12].

2 Preliminaries

In the following, we define our notation (§2.1) and the ad-hoc scenario (§2.2), and give an overview of oblivious transfer (§2.3) and the GMW protocol (§2.4). We describe Yao's garbled circuits in Appendix §A.

2.1 Notation

We denote the two parties that participate in the secure computation as $\mathcal{A}$ and $\mathcal{B}$. We use the standard notation for bitwise operations, i.e., $x \oplus y$ denotes bitwise XOR, $x \wedge y$ bitwise AND, and $x||y$ the concatenation of two bit strings x and y. We refer to the symmetric security parameter as t and the function to be evaluated as f.

2.2 Ad-Hoc Scenario

In an *ad-hoc* secure two-party computation scenario, two parties that do not necessarily know each other in advance want to spontaneously perform secure computation of an arbitrary function f on their private inputs x and y. Traditionally, secure computation protocols consist of two interactive phases: the *setup phase* (independent of x and y) and the *online phase*. We extend this setting by a local *init phase* as depicted in Fig. 1.

The init phase takes place at any time before the parties have identified each other and is used for pre-processing.

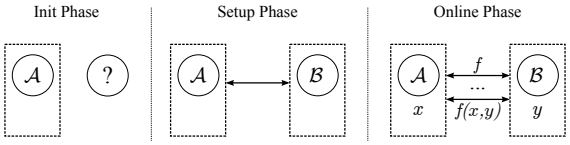

Figure 1: The three secure computation phases.

In the setup phase, the parties have determined their communication partner, establish a communication channel, and know an upper bound on the function size $|f|$. In the online phase, the parties provide their private inputs x and y to the function f that they want to evaluate and begin the secure computation. The *ad-hoc time* is the combined time for setup and online phase.

2.3 Oblivious Transfer

Oblivious transfer (OT) is a fundamental building block for secure computation. In an OT protocol [Rab81], the sender inputs two strings (s_0, s_1). The receiver inputs a bit $c \in \{0,1\}$ and obtains s_c as output without revealing to the sender which of the two messages he chose and without the receiver learning any information about s_{1-c}. OT protocols, such as [NP01], require public-key cryptography and make OT a relatively costly operation. OT extension [IKNP03] allows to increase the efficiency of OT by extending a small number of t base OTs to a large number $n \gg t$ of OTs whilst only using $\mathcal{O}(n)$ symmetric cryptographic operations. Optimizations to the OT extension protocol of [IKNP03] were suggested in [ALSZ13], which allow the parties to reduce the amount of data sent per OT. Moreover, [ALSZ13] describes a more efficient variant of the OT extension protocol for computing random OT, where the sender obtains two random values as output of the OT protocol.

2.4 The GMW Protocol

In the GMW protocol [GMW87], two (or more) parties compute a function f, represented as Boolean circuit on their private inputs by secret sharing their inputs using an XOR secret sharing scheme and evaluating f gate by gate. Each party can evaluate XOR gates locally by computing the XOR of the input shares. AND gates, on the other hand, require the parties to interact with each other by either evaluating an OT or by using a *multiplication triple* [Bea91] as shown in Appendix §B. Finally, all parties send the shares of the output wires to the party that shall obtain the function output. The main cost factors in GMW are the total number of AND gates in the circuit, called (multiplicative) size $|f|$, and the highest number of AND gates between any input wire and any output wire, called (multiplicative) depth $d(f)$.

Because an interactive OT is required for each AND gate, it was believed that GMW is very inefficient compared to Yao's garbled circuits. However, in [CHK+12] it was shown that by using OT extension [IKNP03] and OT pre-computation [Bea95] many OTs can be pre-computed efficiently in an interactive setup phase. Thereby, all use of symmetric cryptographic operations is shifted to the setup phase, leaving only efficient one-time pad operations for the online phase. Additionally, the setup phase only requires an upper bound on $|f|$ to be known before the secure computation. Follow-up work of [SZ13] demonstrated that, by using OT to pre-compute multiplication triples in the setup phase, the online phase can be further sped up. Multiplication triples are random-looking bits a_i, b_i, and c_i, for $i \in \{\mathcal{A}, \mathcal{B}\}$, satisfying $(c_\mathcal{A} \oplus c_\mathcal{B}) = (a_\mathcal{A} \oplus a_\mathcal{B}) \wedge (b_\mathcal{A} \oplus b_\mathcal{B})$, that are held by the respective parties and used to mask private data during the secure computation. This masking is done very efficiently, since no cryptographic operations are required. In [ALSZ13] it was shown that multiplication triples can be generated interactively using two random OTs. [Hua12] proposed to let a trusted server generate the multiplication triples and send (a_i, b_i, c_i) to party i over a secure channel via the Internet. In our work, we propose to do this locally, without knowing the communication partner in advance.

3 Our Setting

In our setting, depicted in Fig. 2, we focus on efficient ad-hoc secure computation between two semi-honest (cf. §3.1) parties $\mathcal{A}$ and $\mathcal{B}$ who each hold a mobile device, which are approximately equally powerful but significantly weaker than typical desktop computer systems. The parties' devices are connected via a wireless network and are battery-powered.

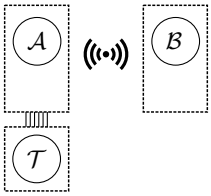

Figure 2: The parties involved in the secure computation.

$\mathcal{A}$ holds a general-purpose tamper-proof hardware token $\mathcal{T}$ that has very few computational resources. $\mathcal{T}$ is powered by $\mathcal{A}$, and its functionalities are limited to the standard functionalities described in §3.2. $\mathcal{A}$ and $\mathcal{T}$ are connected via a physical low-bandwidth connection and communicate via a fixed interface. $\mathcal{B}$ and $\mathcal{T}$ communicate via $\mathcal{A}$, i.e., every message that $\mathcal{B}$ and $\mathcal{T}$ exchange, is seen and relayed by $\mathcal{A}$. Note that this directly requires all communication between $\mathcal{B}$ and $\mathcal{T}$ to be encrypted such that it cannot be read by $\mathcal{A}$. We assume that $\mathcal{T}$ behaves semi-honestly, and is issued by a third party, external to and trusted by both $\mathcal{A}$ and $\mathcal{B}$ (cf. §3.2).

3.1 Adversary Model

We assume that both parties behave semi-honestly in the online phase, i.e., they follow the secure computation protocol, but may try to infer additional information about the other party's inputs from the observed messages. To the best of our knowledge, all previous work on secure computation between two mobile phones is based on the semi-honest model (cf. §7.1). The semi-honest model is suitable in scenarios where the parties want to prevent inadvertent information leakage and for devices where the software is controlled by a trusted party (e.g., business phones managed by an IT department) or where code attestation can be applied. Moreover, this model gives an estimate on the achievable performance of secure computation. We outline how to extend our protocol to malicious security in Appendix §F.

3.2 Trusted Hardware Token

We use the term trusted hardware token $\mathcal{T}$ to refer to a tamper-proof, programmable device, such as a Java smartcard, that offers a restricted set of functionalities. Such functionalities include, for instance, hashing, symmetric and asymmetric encryption/decryption, secure storage of private keys, and secure random number generation. A detailed summary of standard smartcard functionalities is given in [HL08]. The hardware token is passive, i.e., it cannot initiate a communication by itself and only responds to queries from its host. It contains both persistent and transient memory. $\mathcal{T}$ is physically protected against attacks and is securely erased if it is opened by force. Each token holds an asymmetric key pair, similar to an endorsement key used in TPMs [TCG13], where the public key is certified by a known trusted third party and allows unique identification of $\mathcal{T}$.

Tiny Trusted Third Party $\mathcal{T}$ acts as a tiny trusted third party that behaves semi-honestly. This assumption is similar to the TPM model that is widely used in desktop environments. $\mathcal{T}$ only provides correlated randomness that is later used in the secure computation and does never receive any of $\mathcal{A}$ or $\mathcal{B}$'s private inputs. We assume that only certified code is allowed to be executed on $\mathcal{T}$, and that $\mathcal{T}$ can only actively deviate from the protocol if the hardware token's manufacturer programmed it to be malicious. We assume the code certification was carried out by a trusted third party, and argue that both the manufacturer and the certification authority would face severe

reputation loss if it was discovered that they built backdoors into their products. Moreover, we assume that neither $\mathcal{A}$ nor $\mathcal{B}$ colludes with the hardware token manufacturer. This non-collusion assumption is a common requirement for outsourced secure computation schemes such as [Hua12, KMR12, CMTB13] and enables the construction of efficient protocols. Finally, note that, although $\mathcal{T}$ is in $\mathcal{A}$'s possession, $\mathcal{A}$ cannot easily corrupt $\mathcal{T}$ or obtain its internal information, since $\mathcal{T}$ is assumed to be tamper-proof and does not reveal internal secrets, i.e., the costs of an attack are higher than the benefits from breaking $\mathcal{T}$'s security. This assumption also holds if $\mathcal{A}$ colludes with or impersonates $\mathcal{B}$.

Protection Against Successful Hardware Attacks A malicious adversary could try to break into the hardware token. If such an attack is successful, the following standard countermeasures can be used to prevent further damage: A binding between token and key pair can be realized by using techniques such as physically uncloneable functions (PUFs), however, we are not aware of solutions that are available in commercial products. To bind a token to a certain mobile device or person, $\mathcal{T}$'s certificate could be personalized with one or multiple values that are unique per user and that can be verified over an off-band channel, such as the user's telephone number or the ID of the user's passport. Another line of defense can be certificate revocation lists (CRLs) that allow the users to check if a token is known to be compromised or malicious.

4 Token-aided Mobile GMW

In the following section, we give details on our token-aided GMW-based protocol on mobile devices. Our goal is to minimize the ad-hoc time, i.e., the time from establishing the communication channel between $\mathcal{A}$ and $\mathcal{B}$ until receiving the results of the secure computation. We consider the init phase to not be time critical, but we try to keep its computational overhead small.

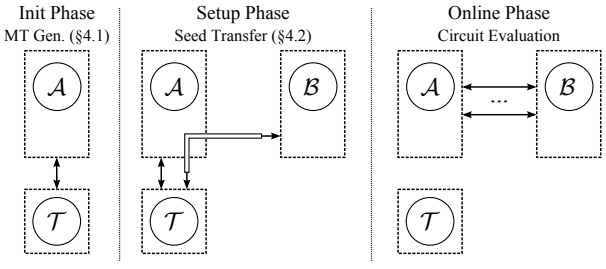

Figure 3: The three phases, workload distribution, and communication in our token-aided scheme.

An overview of our protocol is given in Fig. 3. The general idea is to let the hardware token generate multiplication triples from two (or more) seeds in the init phase that are independent of the later communication partner (§4.1). In the setup phase, $\mathcal{T}$ then sends one seed to $\mathcal{A}$ and the other seed over an encrypted channel to $\mathcal{B}$ (§4.2). The token thereby replaces the OT protocol in the setup phase and allows pre-computing the multiplication triples independently of the communication partner. The online phase of the GMW protocol remains unchanged. In order to overcome the restriction that the function size needs to be known in advance, we describe a method that pre-computes several multiplication triple sequences of different size and only adds a small communication overhead in the setup phase (§4.3). Finally, we analyze the security of our protocol (§4.4) and compare its performance to previous solutions (§4.5).

4.1 Multiplication Triple Pre-Computation in the Init Phase

In the original GMW protocol, $\mathcal{A}$ and $\mathcal{B}$ interactively compute their multiplication triples $(a^n_\mathcal{A}, b^n_\mathcal{A}, c^n_\mathcal{A})$ and $(a^n_\mathcal{B}, b^n_\mathcal{B}, c^n_\mathcal{B})$ in the setup phase using $2n$ random OT extensions (cf. §2.4). Instead, we avoid this overhead in the setup phase and let $\mathcal{T}$ pre-compute the multiplication triples in the init phase as shown in Fig. 4: $\mathcal{T}$ first *generates* random seeds and then *expands* these seeds internally into the multiplication triples and sends $c^n_\mathcal{A}$ to $\mathcal{A}$.

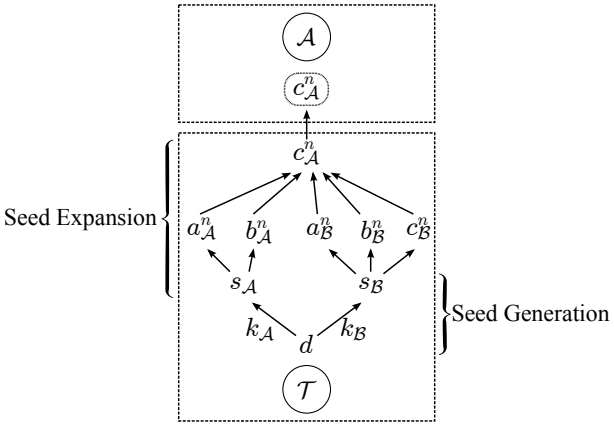

Figure 4: Multiplication triple pre-generation in the init phase between $\mathcal{A}$ and $\mathcal{T}$.

Seed Generation In the seed generation step, $\mathcal{T}$ generates two seeds $s_\mathcal{A} = G_{k_\mathcal{A}}(d)$ and $s_\mathcal{B} = G_{k_\mathcal{B}}(d)$ using a cryptographically strong Pseudo-Random Generator (PRG) G, two master keys $k_\mathcal{A}$ and $k_\mathcal{B}$, and a state value d, which is unique per multiplication triple sequence and can be instantiated with a counter. The two master keys $k_\mathcal{A}$

and $k_{\mathcal{B}}$ are constant for all multiplication triple sequences and have to be generated and stored only once. Thereby, $\mathcal{T}$ has to store only the unique state value d in its internal memory for every multiplication triple sequence. Note that the only values that will leave the internal memory of $\mathcal{T}$ are the seeds $s_{\mathcal{A}}$ and $s_{\mathcal{B}}$ that will be sent in the setup phase to $\mathcal{A}$ and $\mathcal{B}$, respectively (cf. §4.2). In order to ensure that $s_{\mathcal{B}}$ is not sent out twice, we require $s_{\mathcal{A}}$ to be queried before $s_{\mathcal{B}}$ and delete the state value d as soon as $s_{\mathcal{B}}$ has been sent out over the encrypted channel. A security analysis of this scheme is given in §4.4.

Seed Expansion The seed expansion step computes a valid multiplication triple sequence from the seeds $s_{\mathcal{A}}$ and $s_{\mathcal{B}}$ by computing $(a^n_{\mathcal{A}}, b^n_{\mathcal{A}}) = G_{s_{\mathcal{A}}}(d_{\mathcal{A}})$ and $(a^n_{\mathcal{B}}, b^n_{\mathcal{B}}, c^n_{\mathcal{B}}) = G_{s_{\mathcal{B}}}(d_{\mathcal{B}})$ and setting the remaining value $c^n_{\mathcal{A}} = (a^n_{\mathcal{A}} \oplus a^n_{\mathcal{B}}) \wedge (b^n_{\mathcal{A}} \oplus b^n_{\mathcal{B}}) \oplus c^n_{\mathcal{B}}$, where $d_{\mathcal{A}}$ and $d_{\mathcal{B}}$ are publicly known state values of $\mathcal{A}$ and $\mathcal{B}$, respectively. Due to the limited memory of the hardware token, the sequence $c^n_{\mathcal{A}}$ is computed block-wise such that $\mathcal{T}$ requires only a fixed amount of memory, independently of n, and each block is sent to $\mathcal{A}$, who stores it locally. Note that the values $(a^n_{\mathcal{A}}, b^n_{\mathcal{A}}, a^n_{\mathcal{B}}, b^n_{\mathcal{B}}, c^n_{\mathcal{B}})$ do not need to be stored, since they can be expanded from $s_{\mathcal{A}}$ and $s_{\mathcal{B}}$, respectively.

4.2 Seed Transfer in the Setup Phase

In the setup phase, the hardware token sends the seeds $s_{\mathcal{A}}$ and $s_{\mathcal{B}}$ to $\mathcal{A}$ and $\mathcal{B}$, respectively, and the parties generate their multiplication triples as depicted in Fig. 5. $\mathcal{A}$ obtains his seed $s_{\mathcal{A}}$ directly from $\mathcal{T}$ and can read the sequence $c^n_{\mathcal{A}}$, which was obtained in the init phase, from its internal flash storage. $\mathcal{B}$'s seed $s_{\mathcal{B}}$, on the other hand, cannot be sent in plaintext from $\mathcal{T}$ to $\mathcal{B}$ as the communication between the token and $\mathcal{B}$ is relayed over $\mathcal{A}$, which would allow $\mathcal{A}$ to intercept $s_{\mathcal{B}}$ and thus break the security of the scheme. We therefore require the communication between $\mathcal{B}$ and $\mathcal{T}$ to be encrypted and $\mathcal{T}$ to authenticate itself to $\mathcal{B}$ with a certificate.

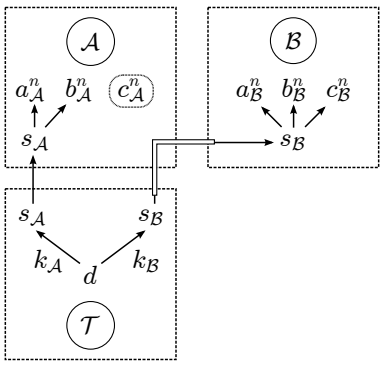

Figure 5: Seed transfer and seed expansion in the setup phase. $s_{\mathcal{B}}$ is sent from $\mathcal{T}$ to $\mathcal{B}$ over a secure channel.

An encrypted and one-way authenticated communication channel can be established using a key agreement protocol from a wide variety of choices (cf. [MvOV96]). We choose two protocols that allow us to handle different attacker models: For security against a malicious (active) $\mathcal{A}$ we use *TLS* [IET08] (with RSA for public-key crypto, AES for symmetric encryption, and HMAC as message authentication code) and for security against a semi-honest (passive) $\mathcal{A}$ we use KAS1-basic [NIS09] (with AES for symmetric encryption), cf. Appendix §C for details. Both schemes use $\mathcal{T}$'s public-key certificate that is signed by a trusted third party. For every new connection this certificate is verified by $\mathcal{B}$ and optionally checked against a CRL and/or is checked to be consistent with $\mathcal{A}$'s identity over an out-of-band channel to protect against successful hardware attacks (cf. §3.2).

4.3 Multiplication Triple Composition

The multiplication triple generation described until now requires the function size $n = |f|$ to be known beforehand. While this may be the case for some functions, e.g., for set intersection using bitwise AND (cf. §6.1), the size of other functions depends on the number of inputs, e.g., the number of contacts in the address book (cf. §6.3). The naive solution to not knowing n in advance would be to generate several multiplication triple sequences of fixed size ℓ in the init phase and send their $\lceil n/\ell \rceil$ seeds in the setup phase, when n is known. However, on average this approach wastes $\ell/2$ multiplication triples and requires to send $\lceil n/\ell \rceil$ multiplication triple seeds. Thus, a smaller ℓ results in fewer wasted multiplication triples but more communication overhead, while a higher ℓ results in more wasted multiplication triples but less communication. Since typical function sizes in secure computation range from millions [HEKM11] to even a billion AND gates [CMTB13], an appropriate ℓ is difficult to choose.

Instead of generating fixed-length blocks of multiplication triple sequences, we propose to generate m multiplication triple sequences $s_0, \ldots, s_{m-1}$ in the init phase, where s_i contains 2^i $(0 \leq i < m)$ multiplication triples. In the setup phase, we then send a set of multiplication triple seeds $\{s_k \,|\, n_k = 1\}$, where n_k is the k-th bit of n. This approach requires sending at most $\lceil \log_2 n \rceil$ seeds. As communication between $\mathcal{T}$ and $\mathcal{A}$ is the bottleneck in our implementation, we set the smallest size of a multiplication triple sequence such that it fits into one packet.

4.4 Security Analysis

In this section, we briefly analyze the security of our protocol for each of the three secure computation phases.

Init Phase In the init phase no private inputs are involved and $\mathcal{B}$ is unknown. Therefore, $\mathcal{A}$ can only try to manipulate the token, which is hard since the hardware token is tamper-proof. Moreover, $\mathcal{A}$ receives only its $c_{\mathcal{A}}$ shares that do not reveal anything about $\mathcal{B}$'s shares or $\mathcal{T}$'s internal state, due to the cryptographically strong PRG.

Setup Phase The only attack a malicious $\mathcal{A}$ could play in the setup phase, is to impersonate $\mathcal{B}$. This attack is prevented, since every seed s_B can only be queried once (cf. §4.1). The communication between the hardware token and $\mathcal{B}$ is done through an encrypted channel, so that $\mathcal{A}$ cannot get access to those messages. For active security, we use TLS and add a MAC to every packet to prevent modifications and avoid replay attacks. $\mathcal{B}$ cannot actively attack the token since all communication to the hardware token is controlled by $\mathcal{A}$. Obviously, any party can drop or ignore messages, but we exclude this simple denial of service attack from our system model since we assume both parties to be willing to participate in the secure computation. The seeds that each party obtains from the hardware token do also not reveal any additional information since they are directly output from a cryptographically strong PRG to which the hardware token's internal state is used as seed.

Online Phase The security for the online phase directly carries over from the GMW protocol, as we do not introduce any modifications to this phase.

4.5 Performance Comparison

We show that the asymptotic performance of our protocol improves over existing solutions. A summary is shown in Tab. 1 on page 2 and a more detailed comparison is given in Appendix §D. An experimental evaluation of our protocol is provided in §5.4 and its performance on applications is evaluated in §6.

Asymptotic Performance The init phase of our protocol is, unlike [JKSS10], independent of a concrete instance of f and can thus be pre-computed without knowing a communication partner. During the setup phase, the communication complexity of our protocol is only $O(t)$ (or $\mathcal{O}(t \cdot \log_2 |f|)$ if $|f|$ is unknown), which improves upon the communication of Yao's protocol and the GMW protocol [CHK+12, SZ13, ALSZ13] with $\mathcal{O}(t \cdot |f|)$ communication. Both parties have to do $\mathcal{O}(|f|/b)$ symmetric cryptographic operations to expand their seeds.[2] The online phase is the first phase where f needs to be known. Here, $\mathcal{A}$ and $\mathcal{B}$ send $\mathcal{O}(|f|)$ bits in $d(f)$ rounds, where $d(f)$ is the depth of f. The parties' computation complexity is negligible, as no cryptographic operations are evaluated. This is the biggest advantage over Yao's garbled circuits protocol [MNPS04, JKSS10, HCE11], where $\mathcal{O}(|f|)$ symmetric cryptographic operations have to be evaluated during the online phase.

Concrete Performance For 80 bit security, the best known instantiation of Yao's garbled circuits protocol (resp. the GMW protocol) require per AND gate 240 bit (resp. 164 bit) communication and $4+1$ (resp. $12+0$) evaluations of symmetric cryptographic primitives in the setup + online phase. In comparison, our solution requires only 4 bit communication and $0.04+0$ fixed-key AES operations per AND gate.

5 Implementation

This section details the implementation of our scheme. We introduce the smartcard that we use to instantiate the hardware token (§5.1), give an overview of our Android implementation (§5.2), outline our benchmarking environment (§5.3), and experimentally compare the OT extension-based multiplication triple generation to our hardware token-based protocol (§5.4).

5.1 G&D Mobile Security Card

In our implementation we instantiate the trusted hardware token $\mathcal{T}$ with the Giesecke & Devrient (G&D) Mobile Security Card SE 1.0 (MSC). It is embedded into a microSD card that additionally contains 2 GB of separate flash memory. The MSC is based on an NXP SmartMX P5CD080 micro-controller that runs at a maximum frequency of 10 MHz, has 6 kB of RAM, 80 kB of persistent EEPROM, and is based on Java Card version 2.2.2. Note that an applet can only use 1,750 Bytes of the 6 kB RAM for transient storage. The MSC has co-processors for 3DES, AES and RSA that can be called from a Java Card applet, as well as native routines for MD5, SHA-1 and SHA256. The MSC runs the operating system G&D Sm@rtCafe Expert 5.0 which manages the installed Java Card applets, personalization data, and communication keys. The communication between the Android operating system and the MSC is done by a separate service via the SIMalliance Open Mobile API.

[2] In our implementation, we instantiate the PRG G with AES-128-CTR, which has block size $b = 128$.

5.2 Architecture

The architecture of our implementation is depicted in Fig. 6. To support flexibility and extensibility, our modular architecture consists of the *Application* that specifies the functionality, the *GMWService* that performs secure computation, and the *MTService* that performs the multiplication triple generation and transfer. All communication between $\mathcal{A}$ and $\mathcal{T}$ is done via the MSC Smartcard Service supplied by G&D. The Application can be implemented by a designer and specifies the desired secure computation functionality as a Boolean circuit that can, for instance, be compiled from a high-level circuit description language such as the Secure Function Definition Language (SFDL) [MNPS04, MLB12] or the Portable Circuit Format (PCF) [KMSB13].

The GMWService implements the GMW protocol and performs the secure computation, given a circuit description and corresponding inputs. The MTService generates the multiplication triples using either OT extension (*OT-Ext*) based on the memory efficient implementation of [HS13] including the optimizations from [ALSZ13] or, if one of the parties holds a hardware token, our token-aided protocol of §4. If a hardware token is present, the MTService manages the multiplication triple generation during the init phase by querying the token and storing the received $c_{\mathcal{A}}$ sequences. For the MSC, the multiplication triple generation on $\mathcal{T}$ is performed via a Java Card applet (*MT JC Applet*) that implements the functionality in §4.1 and is accessible through the Java Card interface. Our implementation can be installed as a regular Android app and does not require root access to the smartphone or a custom firmware.

Secure computation is performed by having an Application running on each smartphone, which specifies the function f both parties want to compute securely. From this function the Application generates a circuit description, which it sends to the GMWService. The GMWService interprets the circuit and queries the MTService for the required number of multiplication triples $|f|$. The MTServices on both smartphones then communicate with each other and check whether one of the smartphones holds a hardware token ($\mathcal{A}$ in Fig. 6). If so, both MTServices perform the seed transfer protocol (cf. §4.2), expand the obtained seeds ($\mathcal{A}$ loads the corresponding $c_{\mathcal{A}}$ sequences obtained in the init phase), and merge the obtained multiplication triple sequences (cf. §4.3). If no hardware token is present, the MTServices generate the multiplication triples by invoking OT extension. The MTService then provides the multiplication triples (a_i, b_i, c_i) for $i \in \{\mathcal{A}, \mathcal{B}\}$ to the GMWService. Finally, the Applications send their inputs x and y, respectively, to the GMWService, which performs the secure computation and returns the output $z_i = f(x, y)$.

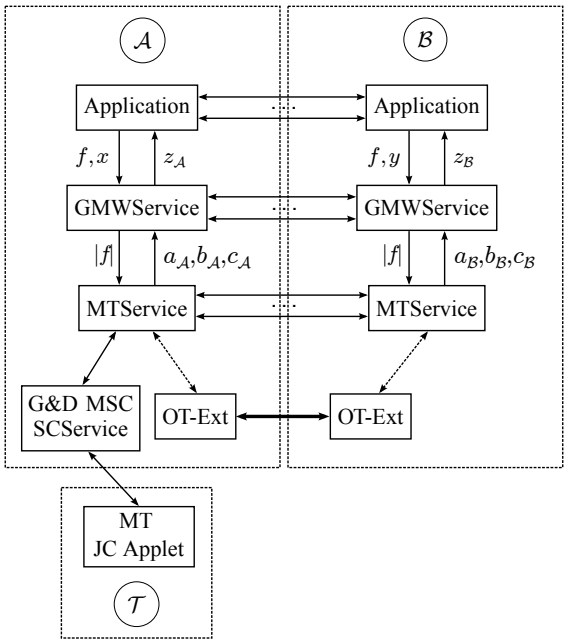

Figure 6: Modular architecture design.

5.3 Benchmarking Environment

For our mobile benchmarking environment we use two Samsung Galaxy S3's, which each have a 1.4 GHz ARM-Cortex-A9 Quad-Core CPU, 1 GB of RAM, 16 GB of internal flash memory, a mircoSD card slot, and run the Android operating system version 4.1.2 (Jelly Bean). For the communication between the smartphones, we use Wi-Fi direct. For the evaluation, we put the smartphones next to each other on a table. The G&D mobile security card is connected to the mircoSD card slot of one of the phones. We use the short-term security setting recommended by NIST [NIS12], i.e., a symmetric key size of 80 bits and a public key size of 1,024 bit with a 160 bit subgroup. We instantiated the pseudo-random generator G that is used for seed expansion (cf. §4.1) with AES-128 in CTR mode. The hardware token generates multiplication triple sequences of size 2^m for $11 \leq m \leq 24$. We used $m = 11$ as lower bound on the size, since 2,048 is the biggest size we can transfer from $\mathcal{T}$ to $\mathcal{A}$ with a single packet, and $m = 24$ as upper bound, since it was appropriate for our case studies in §6. Finally, we point out that our implementation is single-threaded and utilizes only one of the four available cores of our smartphones. We leave the extension to multiple threads as future work.

5.4 Performance Evaluation

First, we want to quantify the runtime differences between the mobile and the desktop environment. We measure the execution time for AES-128 in ECB mode for an identi-

cal single-threaded Java implementation in both domains. The smartphone version is running with 5.5 MB/s while the desktop version achieves 61.1 MB/s. The optimized AES-256 implementation of Truecrypt[3], written in C/C++ and assembly, achieves 143.1 MB/s on the same desktop machine, running without parallelization. For comparison, the smartcard (cf. §5.1) is running AES-128 at a maximum speed of 16.7 KB/s.

In the following we evaluate the performance of our token-based scheme (cf. §4) on smartphones, using TLS or KAS1-basic as key agreement protocol, and compare it to the OT extension based multiplication triple generation. In our evaluations we only include the time for init and setup phase, since the online phase is identical for both approaches. Results for the online phase are given in §6. All values are averaged over 10 measurements.

Fig. 7 gives an overview over the timings for the generation of 2^m ($11 \leq m \leq 24$) multiplication triples using either OT extension in the setup phase or the hardware token (§4.1) in the init phase. Additionally, the setup phase using TLS and KAS1-basic is depicted, which includes the seed transfer and the seed expansion of $\mathcal{B}$. We always assume the worst case number of seeds to be transferred, i.e., for 2^{24} multiplication triples, we transfer $24 - 10 = 14$ seeds (cf. §4.3). Both axes in Fig. 7 are given in a logarithmic scale.

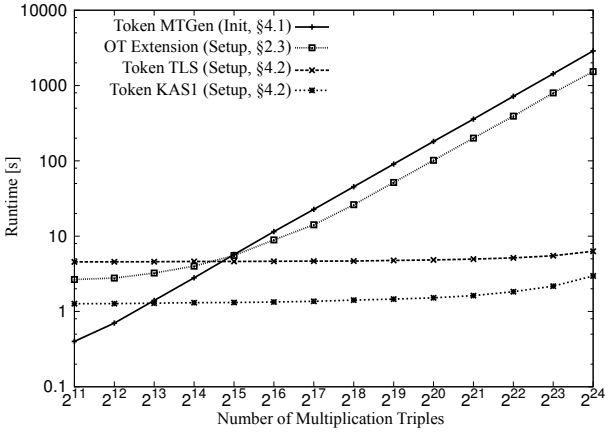

Figure 7: Performance evaluation of the multiplication triple generation and setup phase.

We observe that OT extension on mobile devices is able to generate 2^{24} multiplication triples in 1,529 s, corresponding to 10,971 multiplication triples per second. We ran the same code on two desktop PCs with a 2.5 GHz Intel Core2Quad CPU and 4 GB RAM, connected via Gigabit LAN and were able to compute 2^{24} multiplication triples in 139 s, which indicates a performance decrease of factor 11. While the performance decrease on mobile devices compared to desktop computers was significantly less than the factor of 1000 observed in [HCE11], it is still insufficient for efficiently computing complex functions such as private set-intersection, which typically requires millions of OTsto be evaluated.

In comparison, the multiplication triple generation of the hardware token during the init phase is able to generate 2^{24} multiplication triples in 2,883 s, corresponding to 5,819 multiplication triples per second. For the hardware token-based protocol we observe that the times for sending the seeds using the TLS and KAS1 key agreement protocols grow very slowly with the number of multiplication triples, since the amount of data to be encrypted and sent grows only with $\log_2 |f|$. Additionally, the TLS-based key agreement protocol (4.6 s for 2^{11} multiplication triples) is around factor 3 slower than the KAS1-based key agreement (1.3 s for 2^{11} multiplication triples).

The overall computation and communication workload of OT extension is substantially larger than in our token-based scheme, but its multiplication triple generation rate is not much faster. This can be explained by the faster processing power of the smartphones compared to that of $\mathcal{T}$ and the higher bandwidth of Wi-Fi direct compared to the relatively slow communication channel between $\mathcal{A}$ and $\mathcal{T}$. However, OT extension suffers from high energy consumption, due to the CPU utilization incurred by the symmetric cryptographic operations, as well as the Wi-Fi direct communication [PFW11].

We use PowerTutor[4] to measure the energy consumption of the smartphone's CPU for generating 2^{19} multiplication triples and compare the interactive evaluation of random OT extensions with our smartcard solution. Note that Fig. 8 only displays the CPU's energy consumption whereas the energy consumption of Wi-Fi and the smartcard is not included. However, we argue that the energy consumption of the smartcard is not a critical factor, since these operations can be performed when the phone is charging. The Wi-Fi connection, on the other hand, is required for OT during the setup phase, thus increasing the already high battery drain even further. Moreover, the OT computations have to be done on both devices simultaneously, draining both devices' batteries. Therefore, our token-based solution is particularly well-suited for the mobile domain, where energy consumption and battery lifetime are critical factors.

6 Applications

To evaluate the performance of our protocols, we use the mobile phones and setting as specified in §5.3 and consider the following privacy-preserving applications: availability scheduling (§6.1), location-aware schedul-

[3] http://www.truecrypt.org

[4] http://powertutor.org

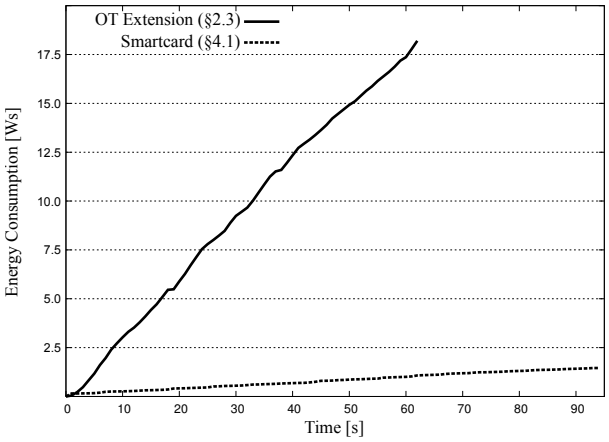

Figure 8: Accumulated smartphone CPU energy consumption during the generation of multiplication triples.

ing (§6.2), and set intersection (§6.3). We implemented the applications and depict the performance results for an average of 10 iterations. We use KAS1-basic (cf. §C) as key authentication scheme. We pre-generated the circuits using the framework of [SZ13], wrote them into a file, and read them on the smartphone. The time for reading the circuit file is included in the setup phase.

6.1 Availability Scheduling

Privacy-preserving availability scheduling is a common example for secure computation on mobile devices [HCC+01, BJH+11] and enables $\mathcal{A}$ and $\mathcal{B}$ to find a possible time slot for a meeting without disclosing their schedules to each other. To schedule a meeting, $\mathcal{A}$ and $\mathcal{B}$ specify a duration and time frame for the meeting. Each party $i \in \{\mathcal{A},\mathcal{B}\}$ then divides the time frame when the meeting can take place (e.g., a week) into n time slots $t_i^n = (t_{i,1},...,t_{i,n})$ and denotes each time slot $t_{i,j} \in \{0,1\}$ as either free ($t_{i,j} = 1$) or occupied ($t_{i,j} = 0$). The parties compute their common availability t_{Avail}^n by computing the bitwise AND of their time slots, i.e., $t_{Avail}^n = t_{\mathcal{A}}^n \wedge t_{\mathcal{B}}^n$. Overall, this circuit has n AND gates and depth 1. Note that the bitwise AND circuit performs a general functionality and can, for instance, be used for privacy-preserving set intersection where elements are taken from a small domain [HEK12] or location matching [CADT13]. For our experiments, we set the time frames s.t. meetings can be scheduled between 8 am and 10 pm for one day divided into 15 minute slots ($n = 56$ slots), one week divided into 15 minute slots ($n = 392$ slots), and one month divided into 10 minute slots ($n = 2{,}604$ slots). We depict our results in the upper half of Tab. 2.

The multiplication triple generation in the init phase can be performed in several hundred milliseconds, since it requires only one (for 56 and 392 time slots) or two (for 2,604 time slots) packet transfers between $\mathcal{T}$ and $\mathcal{A}$. The setup phase, more detailed the seed transfer protocol, is the main bottleneck in this application, as $\mathcal{T}$ has to perform asymmetric and symmetric cryptographic operations. Finally, the online phase requires only milliseconds but has a high variance, due to the communication over Wi-Fi direct and the small number of communication rounds that are performed in the online phase.

For comparison, we evaluated the same circuit using the mobile Yao implementation of [HCE11] on the same phones, which took factor 1.6 (for the day time frame) up to factor 12 (for the month time frame) longer, cf. Tab. 2.

Table 2: Performance for availability and location-aware scheduling. $|f|$ is the size of the circuit and $d(f)$ its depth. All values measured on smartphones (cf. §5.3).

Time Frame	Day	Week	Month
Availability Scheduling §6.1			
$\|f\|$ / $d(f)$	56 / 1	392 / 1	2,604 / 1
Init [s]	0.37 (±1.6%)	0.37 (±1.6%)	0.73 (±1.0%)
Setup [s]	1.3 (±13%)	1.3 (±13%)	1.3 (±13%)
Online [s]	0.002 (±150%)	0.003 (±167%)	0.007 (±129%)
Ad-Hoc [s]	1.3 (±13%)	1.3 (±13%)	1.3 (±13%)
Mobile Yao [HCE11]			
Ad-Hoc [s]	2.14 (±7.1%)	3.82 (±4.7%)	15.9 (±2.7%)
Location-Aware Scheduling §6.2			
$\|f\|$ / $d(f)$	39,864 / 69	280,605 / 87	1,872,206 / 106
Init [s]	6.9 (±0.3%)	48.5 (±0.2%)	319.6 (±0.5%)
Setup [s]	1.4 (±7.1%)	1.8 (±7.0%)	4.8 (±4.8%)
Online [s]	0.16 (±35%)	0.82 (±7.4%)	5.9 (±18%)
Ad-Hoc [s]	1.5 (±8.4%)	2.6 (±6.5%)	10.7 (±11%)

6.2 Location-Aware Scheduling

In the following we show that our system can be adapted to compute arbitrary and complex functions. We introduce the location-aware scheduling functionality which extends the availability scheduling of §6.1, s.t. the distance between the users is considered as well. The location-aware scheduling functionality takes into account the user's location in a time slot, computes the distance between the users, verifies if a meeting is feasible, and outputs the time slot in which the users have to travel the least distance to meet each other. We argue that this approach is practical, since such position information are often already included in the users' schedules.

In the location-aware scheduling scheme, we assume that the user $i \in \{\mathcal{A},\mathcal{B}\}$ also inputs the location of the previous appointment P_i and the next appointment N_i and the distances that he can reach from his previous appointment p_i and from his next appointment n_i (cf. Fig. 9 for an example). Such p_i and n_i can be computed in plaintext using the distance between P_i and N_i, the free time until the next appointment and the duration of the meeting. The minimal distance among all time slots where

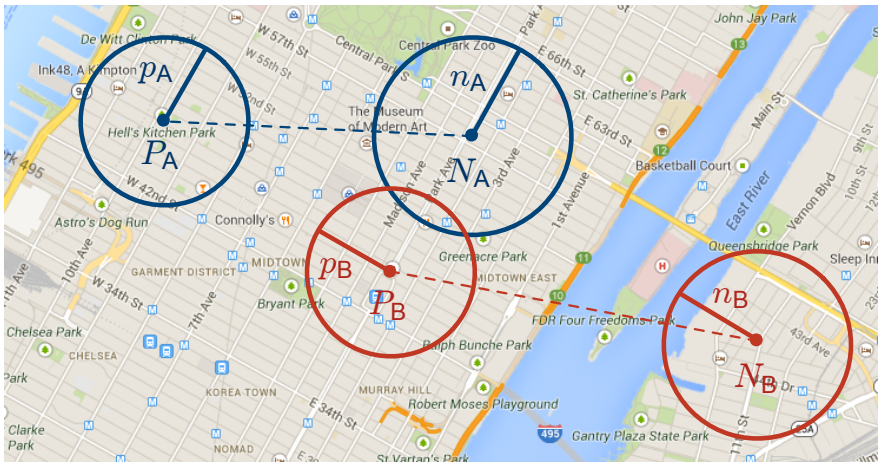

Figure 9: Location-aware scheduling for one time slot of $\mathcal{A}$ and $\mathcal{B}$ with previous locations $P_\mathcal{A}$ and $P_\mathcal{B}$, reachable distances from previous appointments $p_\mathcal{A}$ and $p_\mathcal{B}$, next locations $N_\mathcal{A}$ and $N_\mathcal{B}$ and corresponding reachable distances $n_\mathcal{A}$ and $n_\mathcal{B}$. The meeting can be scheduled between $N_\mathcal{A}$ and $P_\mathcal{B}$ as the reachable ranges overlap.

the reachable ranges for $\mathcal{A}$ and $\mathcal{B}$ overlap is selected as final result. If successful, the function outputs the identified time slot and for each user whether he should leave from the location of the previous or next appointment. A detailed description of the functionality is given in Appendix §E. We evaluate the scheme on the same number of time slots used in §6.1 (day, week, month) and depict the performance in the lower half of Tab. 2.

Compared to availability scheduling, the location-aware scheduling circuit is significantly bigger and requires more communication rounds. When performing the scheduling for a month, the circuit consists of 1.8 million instead of 2,604 AND gates for availability scheduling. The time for the init phase increases linearly with the number of AND gates and requires 319 s when performing scheduling for a month. The time for the setup phase is increased less, since the seed transfer grows only logarithmically in $|f|$ and the seed expansion is done efficiently. The online phase is also slowed down substantially (6 s for a month time frame), but is still practical.

6.3 Private Set Intersection

Private set intersection (PSI) is a widely studied problem in secure computation and can be used for example to find common contacts in users' address books [HCE11]. It enables two parties, each holding a set $S_\mathcal{A}$ and $S_\mathcal{B}$ with elements represented as σ-bit strings to determine which elements both have in common, i.e., $S_\mathcal{A} \cap S_\mathcal{B}$, without disclosing any other contents of their sets. While many special-purpose protocols for PSI exist, e.g., [CT10, CT12, CADT13], generic protocols mostly build on the work of [HEK12], where the Sort-Compare-Shuffle (SCS) circuit was outlined. The idea is to have both parties locally pre-sort their elements, privately merge them, check adjacent values for equality, and obliviously shuffle the resulting values to hide their order.

We implement the SCS-WN circuit of [HEK12] which uses a Waksman permutation network to randomly shuffle the resulting elements. We perform the comparison for bit sizes $\sigma \in \{24, 32, 160\}$ and compare the ad-hoc runtime of our protocol to the implementation of [HCE11] for $\sigma \in \{24, 32\}$. The results from [HEK12] are compared to ours for $\sigma \in \{32, 160\}$. The results are given in Fig. 10 and in Tab. 3. Note that [HCE11] and [HEK12] implement Yao's garbled circuits protocol using pipelining, whereas we use the GMW protocol.

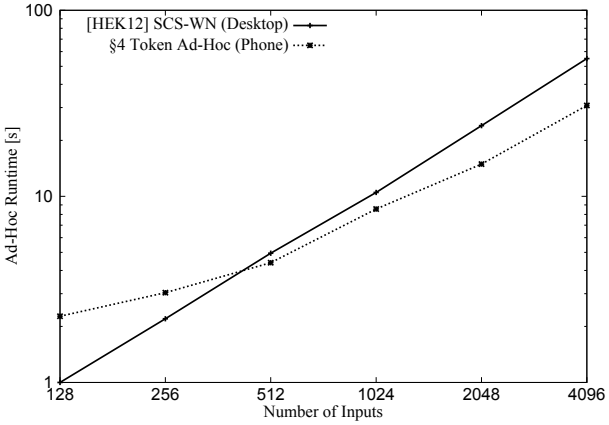

Figure 10: Private set intersection runtime for $\sigma = 32$ bit elements using our token-based protocol on two smartphones (§5.3) and [HEK12] on two desktop PCs.

For a fair comparison, we ran the code from [HCE11] on our Samsung Galaxy S3 smartphones and observed an

approximate speedup of factor 2 compared to the measurements from their paper, that were made on older hardware (two Google Nexus One phones). Note that our performance results, as well as the values for the implementation of [HCE11] are benchmarked on mobile devices connected via Wi-Fi Direct, while [HEK12] is benchmarked on two desktop PCs (two Core2Duo E8400 3GHz PCs connected via 100 Mbps LAN).

From Fig. 10 we observe that, due to the seed transfer in our setup phase (cf. §4.2), the Yao's garbled circuits implementation of [HEK12] is faster for up to 256 inputs. However, the seed transfer time amortizes for larger inputs and our token-based scheme outperforms the implementation of [HEK12], even though our implementation runs on substantially slower mobile phones while theirs is evaluated on two desktop PCs. From Tab. 3 we observe that our scheme outperforms the Yao's garbled circuits implementation of [HCE11], evaluated on identical mobile phones, by factor 18 for 32 inputs with $\sigma = 24$ bit and by up to factor 550 for 1,024 inputs with $\sigma = 32$ bit.

Finally, we compare the performance of our protocol to the PSI protocol of [CADT11, CADT13]. We use their reported numbers for pre-computed PSI on 20 input values and set the bit size $\sigma = 160$ in our protocol.[5] The protocol of [CADT11, CADT13] needs 3.7 s, while our ad-hoc runtime is only 2.1 s ($\pm 4.8\%$). Note, however, that their approach has only a constant number of rounds and can be sped up using multiple cores.

7 Related Work

We classify related work into three categories: secure function evaluation (§7.1), server-aided secure function evaluation (§7.2), and token-based cryptography (§7.3).

7.1 Generic Secure Function Evaluation

The foundations for secure function evaluation (SFE) were laid by Yao [Yao86] and Goldreich et al. [GMW87] who demonstrated that every function that is efficiently representable as Boolean circuit can be computed securely in polynomial time with multiple parties.

SFE Compiler A first compiler for specific secure two-party computation functionalities was presented in [MOR03]. The Fairplay framework [MNPS04] was the first efficient implementation of Yao's garbled circuits protocol [Yao86] for generic secure two-party computation and enabled a user to specify the function to be computed in a high-level language. The FastGC framework [HEKM11] improved on the results of Fairplay by evaluating functions with millions of Boolean gates in mere minutes using optimizations such as the free XOR technique [KS08] and pipelining. The FastGC framework has been used to implement various functions such as privacy-preserving set intersection [HEK12], genomic sequencing, or AES [HEKM11], and was optimized with respect to a low memory footprint in [HS13].

Next to Yao's garbled circuits protocol, the GMW protocol [GMW87] recently received increasing attention. The work of [CHK+12] efficiently implemented GMW in a setting with multiple parties. Subsequently, [SZ13] optimized GMW for the two-party setting and showed that GMW has advantages over Yao's garbled circuits protocol as it allows to pre-compute all symmetric cryptographic operations in a setup phase and that the workload can be split evenly among both parties.

SFE on Mobile Devices A recent line of research aims at making SFE available on mobile devices, such as smartphones. In [HCE11] the authors port the FastGC framework [HEKM11] to smartphones and observe a substantial performance reduction when compared to the desktop environment. They identify the slower processing speed and the high memory requirements as the main bottlenecks. Similarly, [CMSA12] ported the Fairplay framework [MNPS04] to smartphones. A compiler with smaller memory constraints than Fairplay was presented in [MLB12]. We emphasize that previous works on generic SFE on mobile devices use Yao's garbled circuits protocol, whereas our approach is based on GMW.

Several special-purpose protocols for mobile devices using homomorphic encryption were proposed in [BJH+11] (activity scheduling), [CDA11] (scheduling, interest sharing), and [CADT11, CADT13] (comparison, location-based tweets, common friends). In contrast to generic solutions, such custom-tailored protocols can be more efficient, but are restricted to specific functionalities. Their extension to new use cases is complex and usually requires new security proofs.

7.2 Server-Aided SFE

One way to speed up generic secure computations on resource constrained devices is to outsource expensive operations to one or more servers. In [HS12] a system for fair server-aided secure two-party computation using two servers was introduced. SALUS [KMR12] is a system for fair SFE among multiple parties using a single server. A system that allows cloud-aided garbled circuits evaluation between one mobile device and a server was introduced in [CMTB13] and its efficiency was demonstrated on large-scale practical applications, such as a secure path finding algorithm. Both [CMTB13] and [KMR12] achieve security against malicious adversaries, but require

[5] Note that [CADT11, CADT13] also support bigger bit sizes, since they operate on 1,024-bit ElGamal ciphertexts.

Table 3: Ad-hoc runtime of private set intersection where each party inputs n values of σ bits, measured on identical mobile phones (§5.3). [HEK12] results are on PCs and taken from the paper (— indicates that no numbers were given).

Number of Inputs n		32	64	128	256	512	1,024
$\sigma = 24$bit	$\|f\|$	22,432	52,096	118,656	266,240	590,336	1,296,384
	Ours [s]	1.7 (±2.2%)	1.9 (±3.4%)	2.1 (±2.4%)	2.5 (±2.4%)	3.6 (±4.2%)	7.4 (±8.7%)
	[HCE11] [s]	30	68	161	410	1,052	3,010
$\sigma = 32$bit	$\|f\|$	30,368	70,528	160,640	360,448	799,232	1,755,136
	Ours [s]	1.7 (±2.7%)	1.9 (±3.5%)	2.3 (±7.7%)	3.0 (±18%)	4.4 (±9.8%)	8.5 (±20%)
	[HCE11] [s]	42	87	233	565	1,468	4,662
	[HEK12] [s]	—	—	1	2.2	4.95	10.5
$\sigma = 160$bit	$\|f\|$	156,768	364,096	829,312	1,860,864	4,126,208	9,061,376
	Ours [s]	2.2 (±8.8%)	2.7 (±16%)	4.0 (±1.9%)	7.0 (±1.9%)	14.3 (±2.9%)	28.7 (±1.4%)
	[HEK12] [s]	—	—	—	—	—	51.5

at least one party to be a machine with more computing power than a mobile phone as it evaluates multiple garbled circuits. [Hua12] proposes that a trusted server generates multiplication triples that are sent to both parties over a secure channel, requiring $\mathcal{O}(|f|)$ bits communication. Instead, we propose to replace the server with a trusted hardware token and show that the communication to one party can be reduced to sub-linear complexity. Moreover, they achieve security against malicious adversaries based on [NNOB12]; we sketch how to extend our work to malicious security in Appendix §F.

We consider this line of research as orthogonal to ours, since it focuses on outsourcing secure computations to a powerful but untrusted cloud server. In contrast, we focus on secure computation between two mobile devices where computations are outsourced to a trusted, but resource constrained smartcard locally held by one party.

7.3 Token-Based Cryptography

Another approach is to outsource computations to trusted hardware tokens, such as smartcards. These tokens are typically resource-constrained, but have the advantage of offering a tamper-proof trusted execution environment.

Setup Assumptions for UC Hardware tokens can be used as setup assumption for Canetti's universal composability (UC) framework, as they allow to construct UC commitments, with which in turn any secure computation functionality can be realized, e.g., [Kat07, DNW09, DKMQ11]. These works are mainly feasibility results and have not been implemented yet.

SFE in Plaintext As discussed in [HL08], the trivial solution to performing SFE using hardware tokens would be to have each party send its inputs over a secure channel to the token, which evaluates f and returns the output. A similar approach with multiple tokens, which additionally provides fault tolerance was given in [FFP+06].

When using the hardware token for plaintext evaluation, the performance of the time-critical online phase is limited by the performance of the token, which is typically very low. Moreover, this requires the token to hold all input values in memory, which quickly exceeds its very limited resources.[6] Alternatively, the token could use external secure memory to store inputs and intermediate values, e.g., [IS05, IS10], but this would require symmetric cryptographic operations in the online phase. Additionally, each new functionality would have to be implemented on the token, whereas our scheme is implemented only once and supports arbitrary functionalities.

Specific Functionalities An efficient protocol for private set-intersection using smartcards was presented in [HL08]. This protocol was extended to multiple untrusted hardware tokens in [FPS+11]. An anonymous credential protocol was presented in [BCGS09].

Outsourcing Oblivious Transfer There are several works that use hardware tokens to compute oblivious transfer (OT): [GT08] implemented non-interactive OT using an extension of a TPM, [Kol10] proposed OT secure in the malicious model using a stateless hardware token, and [DSV10] provided non-interactive OT in the malicious model using two hardware tokens.

We outsource the setup phase of the GMW protocol, which previously was done via OT, to the hardware token. Previous works on outsourcing n OTs require the hardware token to evaluate $\mathcal{O}(n)$ symmetric (or even asymmetric) cryptographic operations in the ad-hoc phase. In comparison, our scheme requires $\mathcal{T}$ to evaluate $\mathcal{O}(n/t)$ symmetric cryptographic operations in the init phase and only $\mathcal{O}(\log_2 n)$ symmetric cryptographic operations in the setup phase (cf. Tab. 4 on p. 19).

[6] The smartcard we use in our experiments has 1,750 Bytes of RAM, which would be completely filled if each party provided 300 inputs of 24 bits length in private set intersection (cf. §6.3).

8 Conclusion and Future Work

In this work, we demonstrated that generic ad-hoc secure computation can be performed efficiently on mobile devices when aided by a trusted hardware token. We showed how to extend the GMW protocol by such a token, similar to a TPM, to which most costly cryptographic operations can be outsourced. Our scheme pre-computes most of the workload of GMW in an initialization phase, which is performed independently of the later communication partner and without knowing the function or its size in advance. This is particularly desirable as the pre-computation can happen at any time, e.g., when the device is connected to a power source, which happens regularly with modern smartphones. The remaining interactive ad-hoc phase is very efficient and can be executed in a few seconds, even for complex functionalities. We implemented several privacy-preserving applications that are typical for mobile devices (availability scheduling, location-aware scheduling, and set-intersection) on off-the-shelf smartphones using a general-purpose smartcard and showed that their execution times are truly practical. We found that the performance of our scheme is two orders of magnitude faster than that of other generic secure two-party computation schemes on mobile devices and comparable to the performance of similar schemes in the semi-honest adversary model implemented on desktop PCs.

We see several interesting directions for future research. As our scheme is based on the GMW protocol, it can easily be extended to more than two parties, e.g., for securely scheduling a meeting, cf. [CHK+12]. Moreover, our scheme can be modified to also provide security against malicious parties, cf. [Hua12] (we provide more details in Appendix §F). Another direction might be equipping both mobile devices with a hardware token to further improve efficiency and/or security.

Acknowledgements

We thank the anonymous reviewers of USENIX Security 2014 for their helpful comments on our paper. We also thank Giesecke & Devrient for providing us with multiple smartcards and the authors of [HCE11] for sharing their code with us. This work was supported by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE, by the Hessian LOEWE excellence initiative within CASED, and by the European Union Seventh Framework Program (FP7/2007-2013) under grant agreement n. 609611 (PRACTICE).

References

[ALSZ13] G. Asharov, Y. Lindell, T. Schneider, M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Computer and Communications Security (CCS'13)*, p. 535–548. ACM, 2013.

[BCGS09] P. Bichsel, J. Camenisch, T. Groß, V. Shoup. Anonymous credentials on a standard Java card. In *Computer and Communications Security (CCS'09)*, p. 600–610. ACM, 2009.

[Bea91] D. Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – CRYPTO'91*, volume 576 of *LNCS*, p. 420–432. Springer, 1991.

[Bea95] D. Beaver. Precomputing oblivious transfer. In *Advances in Cryptology – CRYPTO'95*, volume 963 of *LNCS*, p. 97–109. Springer, 1995.

[BHKR13] M. Bellare, V. Hoang, S. Keelveedhi, P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *Symposium on Security and Privacy (S&P'13)*, p. 478–492. IEEE, 2013.

[BJH+11] I. Bilogrevic, M. Jadliwala, J.-P. Hubaux, I. Aad, V. Niemi. Privacy-preserving activity scheduling on mobile devices. In *ACM Data and Application Security and Privacy (CODASPY'11)*, p. 261–272. ACM, 2011.

[CADT11] H. Carter, C. Amrutkar, I Dacosta, P. Traynor. Efficient oblivious computation techniques for privacy-preserving mobile applications. Technical report, Georgia Institute of Technology, 2011.

[CADT13] H. Carter, C. Amrutkar, I. Dacosta, P. Traynor. For your phone only: Custom protocols for efficient secure function evaluation on mobile devices. *Journal of Security and Communication Networks (SCN)*, 2013.

[CDA11] E. D. Cristofaro, A. Durussel, I. Aad. Reclaiming privacy for smartphone applications. In *Pervasive Computing and Communications (PerCom'11)*, p. 84–92. IEEE, 2011.

[CHK+12] S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, D. Rubenstein. Secure multi-party computation of Boolean circuits with applications to privacy in on-line marketplaces. In *Cryptographers' Track at the RSA Conference (CT-RSA'12)*, volume 7178 of *LNCS*, p. 416–432. Springer, 2012.

[CMSA12] G. Costantino, F. Martinelli, P. Santi, D. Amoruso. An implementation of secure two-party computation for smartphones with application to privacy-preserving interest-cast. In *Privacy, Security and Trust (PST'12)*, p. 9–16. IEEE, 2012.

[CMTB13] H. Carter, B. Mood, P. Traynor, K. Butler. Secure outsourced garbled circuit evaluation for mobile phones. In *USENIX Security'13*, p. 289–304. USENIX, 2013.

[CT10] E. De Cristofaro, G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security (FC'10)*, volume 6052 of *LNCS*, p. 143–159. Springer, 2010.

[CT12] E. De Cristofaro, G. Tsudik. Experimenting with fast private set intersection. In *Trust and Trustworthy Computing (TRUST'12)*, volume 7344 of *LNCS*, p. 55–73. Springer, 2012.

[DKMQ11] N. Döttling, D. Kraschewski, J. Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In *Theory of Cryptography Conference (TCC'11)*, volume 6597 of *LNCS*, p. 164–181. Springer, 2011.

[DNW09] I. Damgård, J. B. Nielsen, D. Wichs. Universally composable multiparty computation with partially isolated parties. In *Theory of Cryptography Conference (TCC'09)*, volume 5444 of *LNCS*, p. 315–331. Springer, 2009.

[DSV10] M. Dubovitskaya, A. Scafuro, I. Visconti. On efficient non-interactive oblivious transfer with tamper-proof hardware. Cryptology ePrint Archive, Report 2010/509, 2010. http://eprint.iacr.org/2010/509.

[DSZ14] D. Demmler, T. Schneider, M. Zohner. Ad-hoc secure two-party computation on mobile devices using hardware tokens. In *USENIX Security'14*. USENIX, 2014.

[FFP+06] M. Fort, F. C. Freiling, L. D. Penso, Z. Benenson, D. Kesdogan. TrustedPals: Secure multiparty computation implemented with smart cards. In *European Symposium on Research in Computer Security (ESORICS'06)*, volume 4189 of *LNCS*, p. 34–48. Springer, 2006.

[FPS+11] M. Fischlin, B. Pinkas, A.-R. Sadeghi, T. Schneider, I. Visconti. Secure set intersection with untrusted hardware tokens. In *Cryptographers' Track at the RSA Conference (CT-RSA'11)*, volume 6558 of *LNCS*, p. 1–16. Springer, 2011.

[GMW87] O. Goldreich, S. Micali, A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Symposium on Theory of Computing (STOC'87)*, p. 218–229. ACM, 1987.

[GT08] V. Gunupudi, S. R. Tate. Generalized non-interactive oblivious transfer using count-limited objects with applications to secure mobile agents. In *Financial Cryptography and Data Security (FC'08)*, volume 5143 of *LNCS*, p. 98–112. Springer, 2008.

[HCC+01] T. Herlea, J. Claessens, D. De Cock, B. Preneel, J. Vandewalle. Secure meeting scheduling with agenTa. In *Communications and Multimedia Security (CMS'01)*, volume 192 of *IFIP Conference Proceedings*, p. 327–338. Kluwer, 2001.

[HCE11] Y. Huang, P. Chapman, D. Evans. Privacy-preserving applications on smartphones. In *Hot topics in security (HotSec'11)*. USENIX, 2011.

[HEK12] Y. Huang, D. Evans, J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Network and Distributed Security Symposium (NDSS'12)*. The Internet Society, 2012.

[HEKM11] Y. Huang, D. Evans, J. Katz, L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security'11*, p. 539–554. USENIX, 2011.

[HL08] C. Hazay, Y. Lindell. Constructions of truly practical secure protocols using standard smartcards. In *Computer and Communications Security (CCS'08)*, p. 491–500. ACM, 2008.

[HS12] A. Herzberg, H. Shulman. Oblivious and fair server-aided two-party computation. In *Availability, Reliability and Security (ARES'12)*, p. 75–84. IEEE, 2012.

[HS13] W. Henecka, T. Schneider. Faster secure two-party computation with less memory. In *Symposium on Information, Computer and*

Communications Security (ASIACCS'13), p. 437–446. ACM, 2013.

[Hua12] Y. Huang. *Practical Secure Two-Party Computation*. PhD dissertation, University of Virginia, 2012.

[IET08] IETF. The Transport Layer Security (TLS) Protocol Version 1.2. Technical report, Internet Engineering Task Force (IETF), 2008.

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO'03*, volume 2729 of *LNCS*, p. 145–161. Springer, 2003.

[IS05] A. Iliev, S. Smith. More efficient secure function evaluation using tiny trusted third parties. Technical report, Dartmouth College, Computer Science, 2005.

[IS10] A. Iliev, S. W. Smith. Small, stupid, and scalable: secure computing with Faerieplay. In *Workshop on Scalable Trusted Computing (STC'10)*, p. 41–52. ACM, 2010.

[JKSS10] K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, T. Schneider. Embedded SFE: Offloading server and network using hardware tokens. In *Financial Cryptography and Data Security (FC'10)*, volume 6052 of *LNCS*, p. 207–221. Springer, 2010.

[Kat07] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *Advances in Cryptology – EUROCRYPT'07*, volume 4515 of *LNCS*, p. 115–128. Springer, 2007.

[KMR12] S. Kamara, P. Mohassel, B. Riva. Salus: a system for server-aided secure function evaluation. In *Computer and Communications Security (CCS'12)*, p. 797–808. ACM, 2012.

[KMSB13] B. Kreuter, B. Mood, A. Shelat, K. Butler. PCF: a portable circuit format for scalable two-party secure computation. In *USENIX Security'13*, p. 321–336. USENIX, 2013.

[Kol10] V. Kolesnikov. Truly efficient string oblivious transfer using resettable tamper-proof tokens. In *Theory of Cryptography Conference (TCC'10)*, volume 5978 of *LNCS*, p. 327–342. Springer, 2010.

[KS08] V. Kolesnikov, T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *LNCS*, p. 486–498. Springer, 2008.

[MLB12] B. Mood, L. Letaw, K. Butler. Memory-efficient garbled circuit generation for mobile devices. In *Financial Cryptography and Data Security (FC'12)*, volume 7397 of *LNCS*, p. 254–268. Springer, 2012.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella. Fairplay – a secure two-party computation system. In *USENIX Security'04*, p. 287–302. USENIX, 2004.

[MOR03] P. MacKenzie, A. Oprea, M. K. Reiter. Automatic generation of two-party computations. In *Computer and Communications Security (CCS'03)*, p. 210–219. ACM, 2003.

[MvOV96] A. Menezes, P. C. van Oorschot, S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[NIS09] NIST. NIST Special Publication 800-56b, Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization). Technical report, National Institute of Standards and Technology (NIST), 2009.

[NIS12] NIST. NIST Special Publication 800-57, Recommendation for Key Management Part 1: General (Rev. 3). Technical report, National Institute of Standards and Technology (NIST), 2012.

[NNOB12] J. B. Nielsen, P. S. Nordholt, C. Orlandi, S. S. Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology – CRYPTO'12*, volume 7417 of *LNCS*, p. 681–700. Springer, 2012.

[NP01] M. Naor, B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium On Discrete Algorithms (SODA'01)*, p. 448–457. Society for Industrial and Applied Mathematics, 2001.

[NPS99] M. Naor, B. Pinkas, R. Sumner. Privacy preserving auctions and mechanism design. In *Electronic Commerce (EC'99)*, p. 129–139. ACM, 1999.

[PFW11] G. P. Perrucci, F. H. P. Fitzek, J. Widmer. Survey on energy consumption entities on the smartphone platform. In *Vehicular Technology Conference (VTC'11)*, p. 1–6. IEEE, 2011.

[Rab81] M. O. Rabin. *How to exchange secrets with oblivious transfer*, TR-81 edition, 1981. Aiken Computation Lab, Harvard University.

[SZ13] T. Schneider, M. Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *Financial Cryptography and Data Security (FC'13)*, volume 7859 of *LNCS*, p. 275–292. Springer, 2013.

[TCG13] TCG. TCG TPM specifications 2.0, 2013. Trusted Computing Group.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science (FOCS'86)*, p. 162–167. IEEE, 1986.

A Yao's Garbled Circuits Protocol

In Yao's garbled circuits protocol, one party, called the creator, generates a Boolean circuit of the function f and garbles the circuit by assigning symmetric keys to the input wires of the circuit and recursively encrypting every possible output wire key of a Boolean gate using its input wire keys. The creator then sends the garbled circuit together with the wire keys that correspond to his input bits to the other party, called evaluator. Both parties run OT s.t. the evaluator obtains the wire keys that correspond to his input bits. The evaluator then decrypts the garbled circuit gate by gate using the input keys to each gate to obtain the output key. Depending on which party is supposed to obtain the output, the evaluator either sends the resulting keys to the creator or the creator provides a mapping from output keys to their plaintext equivalents.

Several improvements have been proposed for Yao's garbled circuits protocol, and today's most efficient techniques provide free XOR gates [KS08], send $3t$ bits per AND gate over the network using the garbled row reduction technique [NPS99], perform four/one fixed-key AES evaluation per AND gate when creating/evaluating the garbled circuit [BHKR13], and allow the garbled circuit generation and evaluation to be pipelined [HEKM11].

Related Work [IS05, IS10] extended Yao's garbled circuits protocol with an efficient secure array access by employing a secure coprocessor that uses the client as secure external storage. Outsourcing Yao's garbled circuits protocol to a hardware token was discussed in [JKSS10], where the authors proposed to let the hardware token generate the garbled circuit, which reduces the communication complexity.

These schemes that utilize hardware tokens to increase the performance of Yao's garbled protocol require one party to evaluate the garbled circuit, which requires $|f|$ symmetric cryptographic operations during the online phase. As outlined by [HCE11], the resource limitation of mobile devices results in substantial performance penalties when evaluating symmetric cryptographic operations.

B Evaluation of AND Gates in GMW

In the GMW protocol, the evaluation of XOR gates does not require communication between the parties, but AND gates must be evaluated interactively. The AND gate evaluation can be realized with an OT protocol to obliviously obtain the gate's output [CHK+12]. Alternatively, the parties can use (pre-generated) multiplication triples, which are random-looking bits a_i, b_i, and c_i, that are held by the respective parties $i \in \{\mathcal{A}, \mathcal{B}\}$ and satisfy $(c_\mathcal{A} \oplus c_\mathcal{B}) = (a_\mathcal{A} \oplus a_\mathcal{B}) \wedge (b_\mathcal{A} \oplus b_\mathcal{B})$. Multiplication triples can be generated in an interactive way using OT [SZ13, ALSZ13], by a trusted server [Hua12], or by a trusted token, which we propose in our work.

The evaluation of an AND gate with inputs x and y, which are shared between the parties as $x = x_\mathcal{A} \oplus x_\mathcal{B}$ and $y = y_\mathcal{A} \oplus y_\mathcal{B}$, is performed as follows: First, the parties mask the shares x_i and y_i with the multiplication triple shares a_i and b_i for $i \in \{\mathcal{A}, \mathcal{B}\}$ and send the masked inputs to their communication partner.

$$\mathcal{A} \to \mathcal{B} : d_\mathcal{A} = x_\mathcal{A} \oplus a_\mathcal{A},\ e_\mathcal{A} = y_\mathcal{A} \oplus b_\mathcal{A}$$
$$\mathcal{B} \to \mathcal{A} : d_\mathcal{B} = x_\mathcal{B} \oplus a_\mathcal{B},\ e_\mathcal{B} = y_\mathcal{B} \oplus b_\mathcal{B}$$

The variables $d = d_\mathcal{A} \oplus d_\mathcal{B}$ and $e = e_\mathcal{A} \oplus e_\mathcal{B}$ are then calculated locally by both parties.

$$d = x_\mathcal{A} \oplus a_\mathcal{A} \oplus x_\mathcal{B} \oplus a_\mathcal{B} = x \oplus a$$
$$e = y_\mathcal{A} \oplus b_\mathcal{A} \oplus y_\mathcal{B} \oplus b_\mathcal{B} = y \oplus b$$

$z_\mathcal{A}$ and $z_\mathcal{B}$ are the private output shares of the AND gate and locally calculated by each party:

$$\mathcal{A} :\ z_\mathcal{A} = de \oplus db_\mathcal{A} \oplus ea_\mathcal{A} \oplus c_\mathcal{A}$$
$$\mathcal{B} :\ z_\mathcal{B} = db_\mathcal{B} \oplus ea_\mathcal{B} \oplus c_\mathcal{B}$$

Correctness By calculating $z_\mathcal{A} \oplus z_\mathcal{B}$ the AND gate output can be reconstructed. Note that $c = c_\mathcal{B} \oplus c_\mathcal{A} = ab$, according to the definition of multiplication triples. Thus $ab \oplus c = 0$, which allows for the removal in the last step.

$$
\begin{aligned}
z_{\mathcal{A}} \oplus z_{\mathcal{B}} &= de \oplus dy_{\mathcal{A}} \oplus ex_{\mathcal{A}} \oplus c_{\mathcal{A}} \oplus dy_{\mathcal{B}} \oplus ex_{\mathcal{B}} \oplus c_{\mathcal{B}} \\
&= (x \oplus a)(y \oplus b) \oplus (x \oplus a)y_{\mathcal{A}} \oplus (y \oplus b)x_{\mathcal{A}} \oplus c_{\mathcal{A}} \\
&\quad \oplus (x \oplus a)y_{\mathcal{B}} \oplus (y \oplus b)x_{\mathcal{B}} \oplus c_{\mathcal{B}} \\
&= (x \oplus a)(y \oplus b) \oplus (x \oplus a)y \oplus (y \oplus b)x \oplus c \\
&= (x \oplus a)(y \oplus b) \oplus (xy \oplus ay) \oplus (yx \oplus bx) \oplus c \\
&= xy \oplus ay \oplus xb \oplus ab \oplus xy \oplus ay \oplus yx \oplus bx \oplus c \\
&= xy \oplus ab \oplus c \\
&= xy
\end{aligned}
$$

C Key Agreement Protocols

The following section gives a more detailed overview of the implemented key agreement protocols that are used in our protocol to secure the channel from $\mathcal{T}$ via $\mathcal{A}$ to $\mathcal{B}$, as described in §4.2.

TLS Creating a one-way authenticated channel is a common problem on the Internet, where a user wants to access a web-service that is provided by an unknown server in an untrusted network. In this scenario, the Transport Layer Security (TLS) protocol [IET08] is the most widely adopted protocol. TLS provides flexibility by specifying different authentication mechanisms and algorithms for each functionality. We instantiate the seed transfer protocol with the TLS variant using a public-key certificate, which is held by $\mathcal{T}$. $\mathcal{B}$ acts as TLS client and is not authenticated, as he does not hold a certificate. We instantiate TLS with RSA-1536-OAEP as asymmetric primitive, AES-128-CBC as symmetric primitive, SHA256 as hash function and HMAC-SHA256 for message authentication.

KAS1 While TLS is the standard for authenticated key agreement on the Internet, the security guarantees and functionalities it provides are more than what is required for a passive (semi-honest) adversary. We therefore outline a second protocol, the key agreement scheme KAS1-basic [NIS09], which is more lightweight but also provides the required security features for the semi-honest adversary model. It lacks the key confirmation that is offered by TLS, however, this does not affect the confidentiality of the protocol as we can exclude active attacks. To agree on a shared secret key using KAS1-basic, $\mathcal{B}$ acts as initiator and $\mathcal{T}$ as responder. $\mathcal{B}$ creates a shared secret Z, encrypts it using $\mathcal{T}$'s public key and sends it to $\mathcal{T}$ who can then decrypt it with its private key. $\mathcal{T}$ replies with a nonce N_V and both parties can afterwards derive the key material from the mutually known Z, N_V, and the parties' identifiers. In our implementation, we choose $\mathcal{T}$'s certificate serial number and the MAC address of the phone's network interface as party identifiers. We instantiate the asymmetric cipher with RSA-1536-OAEP, use AES-128-CBC as symmetric cipher and SHA256 as key derivation function.

D Detailed Performance Estimation

In the following, we theoretically estimate the performance of our protocol without (Ours A §4.1) and with multiplication triple composition (Ours B §4.3) and compare it to the performance of Yao's garbled circuits protocol using pipelining as implemented in [HCE11], Yao's garbled circuits protocol with hardware token extension as described in [JKSS10], and the GMW protocol with random OT extension as implemented in [SZ13, ALSZ13]. An overview of the comparison is given in Tab. 4. We give the computation and communication complexity and the required information to compute a function and divide them into init phase, setup phase and online phase (cf. §2.2). Note that for the evaluation of Yao's garbled circuits protocol, we omit the input key transfer via OT, since it depends on the actual function that is computed and is negligible for many functions [ALSZ13]. We denote f as the evaluated function, $|f|$ as its multiplicative size, $d(f)$ as its multiplicative depth. t is the symmetric security parameter and b is the block size of the cipher (in our implementation we use AES-128 with $t = b = 128$).

Pipelined Yao [HCE11] In the pipelined Yao's garbled circuits protocol, $\mathcal{B}$ acts as circuit garbler and $\mathcal{A}$ as circuit evaluator (cf. §A). Since the pipelining extension is used, the garbled circuit generation and evaluation is performed on-the-fly in the online phase. $\mathcal{B}$ generates and garbles the circuit by encrypting the truth table for each non-linear Boolean gate, which requires evaluating $4|f|$ symmetric cryptographic operations. $\mathcal{B}$ then sends the garbled circuit of size $3t|f|$ bits to $\mathcal{A}$, and $\mathcal{A}$ evaluates the garbled circuit using $|f|$ symmetric cryptographic operations.

Token-Assisted Yao [JKSS10] A token-assisted variant Yao's garbled circuits protocol was described in [JKSS10]. In their scheme, $\mathcal{B}$ prepares a token $\mathcal{T}$ and sends it to $\mathcal{A}$. $\mathcal{T}$ acts on behalf of $\mathcal{B}$, and generates and locally sends the garbled circuit to $\mathcal{A}$. In the setup phase, $\mathcal{B}$ sends its input keys to $\mathcal{A}$ and $\mathcal{A}$ obtains its input keys directly from $\mathcal{T}$. Finally, in the online phase, $\mathcal{A}$ evaluates the garbled circuit.

In the token-assisted protocol of [JKSS10], $\mathcal{T}$ performs the garbling. Since $\mathcal{T}$ is very resource constrained, it cannot hold the entire garbled circuit in memory at a time and therefore has to re-generate the input and output keys each time it garbles a gate. The computation complexity for garbling the circuit is thereby increased to $7|f|$

Table 4: Comparison of Generic SFE Protocols (f: the function to be computed securely, t: symmetric security parameter, b: block size, d: depth, sym: symmetric cryptographic operations). Empty cells correspond to no complexity or prerequisites.

Phase			Pipelined Yao [HCE11]	Token Yao [JKSS10]	GMW OT-ext [SZ13, ALSZ13]	Ours A §4.1	Ours B §4.3
Init		required		f		$\lvert f\rvert$	
		$\mathcal{A}\Leftrightarrow\mathcal{T}$ comm.		$4t\lvert f\rvert$		$\lvert f\rvert$	
		$\mathcal{T}$ comp.		$7\lvert f\rvert$ sym		$5\lvert f\rvert/t$ sym	
Ad-hoc	Setup	required			$\lvert f\rvert$		$\lvert f\rvert$
		$\mathcal{A}/\mathcal{T}\Leftrightarrow\mathcal{B}$ comm.		t	$2t\lvert f\rvert$	t	$t\cdot\log_2\lvert f\rvert$
		$\mathcal{A}$ comp.			$6\lvert f\rvert$ sym	$2\lvert f\rvert/b$ sym	
		$\mathcal{B}$ comp.			$6\lvert f\rvert$ sym	$3\lvert f\rvert/b$ sym	
	Online	required	f		f		
		$\mathcal{A}\Leftrightarrow\mathcal{B}$ comm.	$3\lvert f\rvert$		$4\lvert f\rvert$, $d(f)$ rounds		
		$\mathcal{A}$ comp.	$\lvert f\rvert$ sym		negligible		
		$\mathcal{B}$ comp.	$4\lvert f\rvert$ sym		negligible		

evaluations of symmetric cryptographic primitives, compared to $4|f|$ in standard Yao's garbled circuits protocol. The communication complexity between $\mathcal{A}$ and $\mathcal{T}$ is also increased to $4t|f|$ compared to $3t|f|$ in standard Yao's garbled circuits protocol, since the garbled row reduction technique cannot be applied.

The evaluation in the online phase is the same as in the standard Yao's garbled circuits protocol and requires $|f|$ evaluations of symmetric cryptographic operations by $\mathcal{A}$. The generation, garbling, and transmission from $\mathcal{A}$ to $\mathcal{T}$ can be done in the init phase but requires the computed function f to be known in advance. If f is not known in advance, these steps have to be performed in the ad-hoc phase.

GMW OT Extension [SZ13, ALSZ13] In contrast to Yao's garbled circuits protocol, the GMW protocol (cf. §2.4) allows to shift all evaluations of symmetric cryptographic operations into the setup phase, which is independent of f, only requires $|f|$ to be known in advance, and evenly balances the workload among the parties [CHK+12, SZ13]. Using the OT extension of [ALSZ13], each party has to evaluate $6|f|$ symmetric cryptographic operations in the setup phase. This is slightly higher than in Yao's garbled circuits protocol, which requires $4|f|$ symmetric cryptographic operations for the creator and $|f|$ for the evaluator. The communication complexity on the other hand is slightly lower for GMW and amounts to $2t|f|$ compared to $3t|f|$ for Yao. In the online phase, the parties only evaluate simple one-time pad operations but have to perform $d(f)$ communication rounds, which makes the performance highly dependent on the network's latency.

Ours A §4.1 In our basic token-assisted GMW protocol described in §4.1, $|f|$ has to be known to have $\mathcal{T}$ pre-compute the multiplication triples in the init phase. To generate $|f|$ multiplication triples, the computation complexity of $\mathcal{T}$ is dominated by $5|f|/b$ symmetric cryptographic operations for the seed expansion. Finally, $\mathcal{T}$ sends the expanded multiplication triples to $\mathcal{A}$ which requires $|f|$ bits to be sent from $\mathcal{T}$ to $\mathcal{A}$. In the setup phase, $\mathcal{T}$ has to send the seeds for the multiplication triple sequences to $\mathcal{B}$ which requires t bits of communication, and $\mathcal{A}$ and $\mathcal{B}$ have to expand their seeds using $2|f|/b$ symmetric cryptographic operations for $\mathcal{A}$ and $3|f|/b$ symmetric cryptographic operations for $\mathcal{B}$. Finally, the parties perform the online phase of GMW, which is the same as for the OT extension-based GMW protocol described before.

Ours B §4.3 The token-assisted GMW protocol with multiplication triple composition described in §4.3 allows the init phase to be independent of $|f|$. Compared to the basic token-assisted protocol, the only overhead that is added is an increase of the communication complexity in the setup phase to $t\cdot\log_2|f|$.

E Description of the Location-Aware Scheduling Algorithm

In the location-aware scheduling application described in §6.2, $\mathcal{A}$ and $\mathcal{B}$ each hold a schedule t_i (for $i\in\{\mathcal{A},\mathcal{B}\}$), that is composed of n time slots, where each time slot $t_{i,j}=(a_{i,j},P_{i,j},p_{i,j},N_{i,j},n_{i,j})$ is assigned an availability $a_{i,j}\in\{0,1\}$, a previous location $P_{i,j}\in(\{0,1\}^\sigma\times\{0,1\}^\sigma)$, a reachable distance from the previous location $p_{i,j}\in\{0,1\}^\sigma$, the next location $N_{i,j}\in(\{0,1\}^\sigma\times\{0,1\}^\sigma)$, and a reachable distance from the next location $n_{i,j}\in\{0,1\}^\sigma$,

where σ is the bit size of the coordinates. The scheduler sets $a_{i,j}$ to whether the time slot is available ($a_{i,j} = 1$) or not, $P_{i,j}$ to the location of the previous appointment of i, and $N_{i,j}$ to the location of the next appointment of i. Additionally, it sets $p_{i,j}$ and $n_{i,j}$ to the distances from $P_{i,j}$ and $N_{i,j}$, respectively, that i can reach given the available time left until the next appointment, the time that i has to travel from $P_{i,j}$ to $N_{i,j}$, and the duration of the meeting. Note that the scheduling algorithm can compute all this in plaintext.

To identify a time slot for an appointment, the scheduler securely computes the distances $d_{0,j}, ..., d_{3,j}$ for all combinations of the previous and next meetings: $d_{0,j} = D(P_{\mathcal{A},j}, P_{\mathcal{B},j})$, $d_{1,j} = D(P_{\mathcal{A},j}, N_{\mathcal{B},j})$, $d_{2,j} = D(N_{\mathcal{A},j}, P_{\mathcal{B},j})$, and $d_{3,j} = D(N_{\mathcal{A},j}, N_{\mathcal{B},j})$, where D is a distance function. To check, if the reachable ranges overlap, the scheduler computes $d_{k,j} \leq r_{k,j}$ where $r_{0,j} = p_{\mathcal{A},j} + p_{\mathcal{B},j}$, $r_{1,j} = p_{\mathcal{A},j} + n_{\mathcal{B},j}$, $r_{2,j} = n_{\mathcal{A},j} + p_{\mathcal{B},j}$, and $r_{3,j} = n_{\mathcal{A},j} + n_{\mathcal{B},j}$. Among these possible meetings, the scheduler selects those that are available for both parties ($a_{\mathcal{A},j} \wedge a_{\mathcal{B},j} = 1$) and the one with the minimal distance $d_{k,j}$, $0 \leq k \leq 3$ for all j in order to minimize the distance that both parties have to travel. The scheduler outputs the index j of the optimal time slot t_j as well as for $\mathcal{A}$ whether the shortest distance was computed from $P_{\mathcal{A},j}$ or $N_{\mathcal{A},j}$ and for $\mathcal{B}$ whether the shortest distance was computed from $P_{\mathcal{B},j}$ or $N_{\mathcal{B},j}$. In our experiments we use $\sigma = 16$ bit coordinates and the Manhattan distance as distance function D.

F Extension to Active Security

To achieve security against active adversaries, the TinyOT protocol of Nielsen et al. [NNOB12] can be realized with a trusted hardware token in our mobile setting. Similar to the GMW protocol, TinyOT uses XOR-based secret sharing, such that every bit x is shared amongst the parties $\mathcal{A}$ and $\mathcal{B}$ as $x = x_{\mathcal{A}} \oplus x_{\mathcal{B}}$. The protocol prevents malicious attacks by adding a MAC and local key to every share used in the secure computation and thereby allowing for oblivious authentication. A bit is considered *authenticated* if the authenticity of both shares can be verified by the parties.

[Hua12] has shown that the TinyOT protocol can be based on correlated randomness that is provided by a trusted server. In the following we show how our token-based solution can be extended to realize an efficient pre-computation for the TinyOT protocol while maintaining a low communication between $\mathcal{T}$ and $\mathcal{B}$. The basic idea is to pre-generate all values for $\mathcal{B}$ from a seeded PRF, while $\mathcal{A}$'s values are calculated on $\mathcal{T}$, sent out during the init phase, and stored locally on $\mathcal{A}$. When $\mathcal{B}$ connects to $\mathcal{A}$ to start the secure computation, he receives the seeds for all required values and expands them. Thereby the communication complexity in the ad-hoc phase remains low. In fact, all pre-calculated values can be expanded from a single seed. However, since the global keys Δ_i must be unique and are used in the entire protocol, we cannot apply the trick of §4.3 to compose blocks of multiplication triples of different sizes. Thus, we also require an upper bound for the function size $|f|$ in advance.

F.1 Protocol Description

In the TinyOT protocol, every party $i \in \{\mathcal{A}, \mathcal{B}\}$ holds a unique and uniformly random global key Δ_i that is fixed for one instance of the secure computation protocol. Every bit share x_i, held by party i is authenticated with a MAC M_{x_i} and a uniformly random local key K_{x_i} satisfying $M_{x_i} = K_{x_i} \oplus x_i \Delta_i$, such that x_i and M_{x_i} are held by party i, while Δ_i and K_x are held by the other party. The original TinyOT protocol proposes choosing K_{x_i} at random and calculating M_{x_i}, however, we can also pick a random M_{x_i} and compute $K_{x_i} = M_{x_i} \oplus x_i \Delta_i$ accordingly. Thereby we are able to pre-generate MACs and keys for shares held by both $\mathcal{A}$ and $\mathcal{B}$. The TinyOT protocol allows for local authenticated evaluation of XOR gates and interactive authenticated evaluation of AND gates, which prevents manipulations from malicious adversaries as they can easily be detected.

Init Phase In the init phase, $\mathcal{A}$ initiates the pre-calculation of all required values for the TinyOT protocol on the token. $\mathcal{T}$ calculates $\mathcal{B}$'s values using a PRG and stores the used seed in its secure storage. Values, that are intended for $\mathcal{A}$ are calculated and sent from $\mathcal{T}$ to $\mathcal{A}$, where they are stored locally. The values that are pre-calculated for either party are the global key Δ_i, authenticated random bits with the corresponding local keys and MACs, authenticated AND bits and authenticated OT bits.

Setup Phase After $\mathcal{A}$ and $\mathcal{B}$ connected to each other, a secure channel between the token and $\mathcal{B}$ is established. $\mathcal{T}$ then sends $\mathcal{B}$'s PRG seed to $\mathcal{B}$, where they are expanded. Thereby, we only require a minimal amount of communication between $\mathcal{T}$ and $\mathcal{B}$, for establishing the secure channel and sending one seed.

Online Phase The online phase is identical to the TinyOT protocol without further modifications. Both parties create authenticated shares of their private inputs and begin the secure computation according to the protocol definition. In the end, one or both parties reveal their output shares, validate the MACs, and calculate the plaintext output.