This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

Secure, Fast, and Energy-Efficient Outsourced Authentication for Smartphones

Paolo Gasti, NYIT
Jaroslav Šeděnka, Masaryk University
Qing Yang, College of William and Mary
Gang Zhou, College of William and Mary
Kiran S. Balagani, NYIT

***Abstract*—Common smartphone authentication mechanisms (e.g., PINs, graphical passwords, and fingerprint scans) are not designed to offer security post-login. Multi-modal continuous authentication addresses this issue by frequently and unobtrusively authenticating the user via behavioral biometric signals, such as touchscreen interaction and hand movements. Because smartphones can easily fall into the hands of the adversary, it is critical that the behavioral biometric information collected and processed on these devices is secured. This can be done by offloading encrypted template information to a remote server, and then performing authentication via privacy-preserving protocols.**

In this paper we demonstrate that the energy overhead of current privacy-preserving protocols for continuous authentication is unsustainable on smartphones. To reduce energy consumption, we design a technique that leverages characteristics unique to the authentication setting in order to securely outsource computation to an untrusted Cloud. Our approach is secure against a colluding smartphone and Cloud, thus making it well suited for authentication. We performed extensive experimental evaluation. With our technique, the energy requirement for running an authentication instance that computes Manhattan distance is 0.2 mWh, which corresponds to a negligible fraction of the smartphone's battery capacity. Additionally, for Manhattan distance, our protocol runs in 0.72s and 2s for 8 and 28 biometric features, respectively. We were also able to compute Hamming distance in 3.29s, compared to 95.57s achieved with the previous fastest outsourced computation protocol (Whitewash). These results demonstrate that ours is presently the only technique suitable for low-latency continuous authentication (e.g., with authentication scan windows of 60 seconds or shorter).

I. Introduction

The need for usable, reliable, and secure smartphone user authentication mechanisms is increasing because smartphones routinely access, generate, store, and process users' private information, and because portability and mobility of smartphones intrinsically increases risks of theft and loss. Common smartphone authentication mechanisms offer limited security—simple PINs are easy to guess [11], while strong alphanumeric passwords and swipe patterns are susceptible to attacks from reflections [52], video capture [49], and smudges [2]. Another fundamental limitation of these mechanisms is that they are designed for login-time authentication, and offer no protection against theft or coercion post-login. Although, in principle, it is possible to repeatedly activate the above mechanisms beyond login point, each activation could potentially distract the user, thereby raising usability concerns.

Continuous (or active) authentication mechanisms aim to address post-login authentication by frequently and unobtrusively verifying the user's identity via behavioral biometric signals, such as touchscreen interaction [13], hand movements and gait [12], [50], voice [30], and phone location [48]. However, authenticating smartphone users via behavioral biometric signals raises security and privacy concerns. These signals carry personal identifiable data (*who is the user?*), and expose user information and behavior (*what app is the user accessing? what is the user saying? what is the user's location?*).

Because smartphones can easily fall into the hands of the adversary, it is critical that behavioral biometric signals collected or stored on these devices are secured. Traditionally, biometric signals have been secured using one of the following two approaches: (1) *on-device*, in which a *transformed* version of the template is stored on the smartphone, either using cancelable biometrics [41] or fuzzy commitments [20]; and (2) *off-device*, in which the smartphone authenticates to a remote server [51], [50], thereby *not* storing the template or any biometric information on the device.

At first blush, on-device approaches sound appealing for smartphones because they require limited computation, and no communication with external parties. However, the assumptions on which these approaches base their security guarantees are *not* compatible with smartphones. Cancelable biometrics assume that the transformation applied to the biometric is secret—which is questionable when the adversary can obtain physical access to the smartphone through theft, loss, or coercion. When this assumption fails, these techniques are susceptible to simple correlation and impersonation attacks [34]. Further, cancelable biometrics and fuzzy commitments assume that the underlying biometric has high guessing entropy [4], which is not true in practice [3], [31], [32].

Off-device approaches sidestep these problems by not having to store biometric information on the smartphone. Matching is performed on a remote server, which can additionally implement rate-limiting to mitigate the effects of low guessing entropy. However, in off-device approaches, the server must have access to the biometric template and to the authentication signal in order to authenticate the user. This raises numerous privacy and security issues [19]. The standard way to address these issues is to use a privacy-preserving protocol (e.g., [6], [42], [36]). The inputs to the protocol are the template from the server and a biometric sample from the smartphone. The output is the distance (or the similarity) between the two inputs. Privacy-preserving protocols faithfully (without loss of accuracy) implement the biometric computation, and *provably* guarantee that no additional information is revealed to the server or the smartphone.

Privacy-preserving protocols for authentication must provide

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

provable security against *malicious parties*, i.e., both server and smartphone can arbitrarily deviate from the correct execution of the protocol. In this setting, our experiments (presented in Section IX) show that privacy-preserving authentication adds substantial energy overhead to the authentication pipeline, when implemented using standard protocol construction techniques. In our case, each privacy-preserving authentication attempt (implementing Manhattan distance matching) consumed 275 mWh using garbled circuits [53] with cut-and-choose [17]. At this rate, the battery of a commodity Android smartphone—a Samsung Galaxy S4, in our experiments—can sustain about 35 authentication attempts before being depleted. This means that if the user is authenticated every 60 seconds, the smartphone will run out of battery in approximately half hour, which is clearly unacceptable. In comparison, the energy overhead of continuously collecting behavioral signals for authentication is negligible: accelerometer and gyroscope accounted for a mere 1.6 mWh for 60-second scans, while extracting touchscreen-based features of [44] and [13] used 0.1 mWh. Thus, to make secure privacy-preserving active authentication viable on smartphones, we need new energy-efficient protocol construction techniques.

Contributions. In this paper, we present a new protocol construction technique for reducing energy cost, computation, and communication of privacy-preserving protocols for active authentication on smartphones. Our design offloads most of the smartphone computation to an untrusted outsourcing party (the Cloud), such as Amazon S3 or Microsoft Azure. This leads to both substantial energy savings for the smartphone, and to a dramatic reduction of protocol execution time. Although both garbled circuits and offloading of computation are known techniques, this work brings them together and enhances them in a novel way that results in over 30-fold decrease in protocol execution cost. To our knowledge, this is the first work that implements garbled circuits secure against malicious parties and does not use cut-and-choose.

In order to measure the performance and power consumption of our approach, we implement our protocol on an Android smartphone. Our technique reduces the energy consumption for computation of Manhattan distance to less than 0.2 mWh, and thus has negligible impact on the smartphone's battery life. Our protocol is substantially faster than the state-of-the-art Cloud-aided computation techniques such as Whitewash [8]. For instance, computing the Hamming distance on 1600-bit vectors takes 3.29s with our approach in comparison to 95.57s with Whitewash.

We provide formal proof of security of our construction. Our protocol guarantees privacy (i.e., the parties cannot learn more than what can be inferred from their input and from the protocol's output, regardless of their behavior), and correctness (the protocol either produces a correct output, or it produces no output). To further strengthen the privacy of our approach, the user's biometric template is stored on the server in encrypted form, and the server has no accesses to the decryption key.

Our protocol is secure against a malicious Cloud, even when it colludes with the smartphone. This is important because the Cloud is used to reduce protocol overhead for the smartphone, and is therefore assumed to be paid for and controlled by the party in possession of the smartphone. Therefore, any protocol that assumes non-collusion between the Cloud and the smartphone is potentially vulnerable to attacks.

Organization. We start by presenting the related work in Section II. We review the cryptographic tools used in our construction in Section III. Our system and security models are defined in Section IV. We discuss our approach in Section V, and present a detailed protocol description in Section VI. Formal security proofs of our protocol are presented in Section VII. We compare our protocol to current techniques in Section VIII. Energy, bandwidth, and execution time of our approach are evaluated in Section IX. We conclude in Section X.

II. Background and Related Work

Garbled Circuits. Since the seminal work on garbled circuit evaluation [53], it has been shown that any function can be securely evaluated by representing it as a boolean circuit. Similar results exist for secure evaluation of any function using secret sharing techniques [39], or homomorphic encryption [10].

Recent literature provides optimizations that reduce computation and communication overhead associated with circuit construction and evaluation. Kolesnikov et al. [25] describe a modification that permits XOR gates to be evaluated *for free*, i.e., there is no communication overhead associated with XOR gates, and their evaluation does not involve cryptographic functions. Pinkas et al. [37] additionally give a mechanism for reducing communication complexity of binary gates by 25%. Their work allows each gate to be specified by encoding only three outcomes instead of four. Finally, Kolesnikov et al. [24] improve the complexity of certain commonly used operations such as addition, multiplication, and comparison, by reducing the number of non-XOR gates.

Zahur et al. [54] introduce a technique that allows them to encode AND gates using only two ciphertexts, while still allowing the use of free-XOR [25]. While the size of the resulting garbled circuit decreases by up to 33%, the use of this technique leads to doubling the amount of computation required to evaluate each AND gate. The authors show that this technique leads to a reduction in energy consumption on a desktop computer. However, it is not clear if this translates to a similar reduction on a smartphone device.

Garbled circuits offer security in the semi-honest model. However, a technique called *cut-and-choose* [17] can be used to construct protocols based on garbled circuits, secure in the malicious model. With cut-and-choose, the circuit constructor creates multiple garblings of a circuit. The circuit evaluator randomly selects a subset of these garblings, and asks the server to reveal these circuits' input keys. The circuit evaluator verifies that all circuits are constructed properly, and evaluates the remaining circuits to obtain the result of the computation. Then, both parties switch roles and repeat this process. There are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

several approaches to implement cut-and-choose, each requiring a different number of circuits to achieve a given level of security (see, e.g., [17], [18], [28], and [27]).

Lindell et al. [29] show that it is possible to reduce the cost of cut-and-choose by evaluating multiple circuits in batch. In order to have a probability of undetected errors of 2^{-40}, their technique requires an average of 7.06 circuits when performing 2^{10} evaluations of the same circuit. This is a substantial improvement compared to [27], which requires 40 circuits to achieve the same level of security. Unfortunately, batching cannot be leveraged with active authentication because, for each user, thousands of authentication attempts never occur at once.

Another approach to privacy-preserving computation is fully homomorphic encryption (FHE), first constructed by Gentry [14]. FHE allows computation of arbitrary ring operations in the encrypted domain. Despite advancements in FHE, current implementations are still far too expensive for smartphones (see, for example, [15] and [38]).

Outsourced Computation. Outsourcing computation to an untrusted third party, such as the Cloud, is an effective way to reduce the computation load of one or more protocol participants. This approach is often referred to as Cloud/server-aided computation, or server-assisted cryptography [5]. There exist Cloud-aided protocols designed for specific functionalities (e.g., server-aided private set intersection [21]), as well as generic protocols, which offer security in the presence of malicious adversaries [22], [9], [8]. Next, we provide an overview of three such Cloud-aided protocols that are closely related to our work.

Kamara et al. [22] introduce Salus, a system which includes two server-aided secure function evaluation protocols. Salus efficiently supports an arbitrary number of protocol participants, and is designed to trade computation for communication. In particular, the authors assume that the participants have access to large bandwidth capabilities. This is clearly not the case for smartphones, for which high-bandwidth wireless communication translates to high energy costs. In addition, Salus uses a fair coin tossing protocol to allow the participants to share a random secret key. This further increases communication overhead between the parties. Finally, Salus is secure against malicious server, smartphone, and Cloud, but only as long as at most one party at a time is malicious. This prevents two parties (e.g., the smartphone and the Cloud) to work together against the third (the server), and is therefore known as *non-collusion assumption*.

Carter et al. [9] developed an outsourcing protocol based on the garbled circuit protocol of Kreuter et al. [26]. Similar to Salus, their protocol relies on the Cloud for evaluating garbled circuit. The protocol of Carter et al. is based on outsourced oblivious transfer, which is used to send circuit input labels to the Cloud. Because of lower bandwidth requirements compared to Salus, the protocol is well suited for mobile devices. However, as with Salus, the overall execution time of [9] is still prohibitively long. Moreover, the protocol is secure in the same adversary model as Salus, i.e., it assumes that the parties do not collude.

More recently, Carter et al. [8] introduced Whitewash, a novel secure function evaluation protocol. In contrast to previous work, Whitewash reverses the roles of the parties. In particular, the Cloud is in charge of generating the garbled circuit, while the smartphone simply garbles its own input. Circuit evaluation is performed by the server. The result of this modification is a protocol that is more efficient than previous work, in terms of both execution time and bandwidth. Moreover, Whitewash does not rely on the non-collusion assumption. In fact, it is secure when a malicious smartphone and a malicious Cloud collude—in which case, the security of the protocol is the same as the underlying garbled circuit technique, which is based on Shelat and Shen's protocol [46]. However, because the protocol is based on cut-and-choose, the execution time is very high, even for relatively simple functionalities.

III. Cryptographic Preliminaries

Garbled Circuit Evaluation. Garbled circuits allow two parties (a circuit *constructor*, and a circuit *evaluator*) to securely evaluate any function represented as a boolean circuit. Let κ be the security parameter. Given a circuit composed of gates connected by wires, the circuit constructor "garbles" the circuit by assigning two randomly chosen encryption keys of length κ, denoted as $\omega_{j,0}$ and $\omega_{j,1}$, to each wire j. These keys represent, respectively, 0 and 1. (In the garbled circuits literature, keys are usually referred to as *labels*.) The circuit constructor encrypts each entry of the truth table corresponding to each gate. Values in the truth table are also represented using labels, and each label is encrypted with two keys, ω_{j,b_j} and ω_{l,b_l}, corresponding to the values on the gate's input wires. Therefore, computing the output label of each gate requires knowing two of its input labels, one for each input wire of the gate.

The output of the circuit is encoded in its *output labels*, constructed and interpreted as follows. Let $\omega_{i,b}$ be the label of output wire i corresponding to output bit b, and s the number of output wires. The circuit constructor selects a pair of random labels $\omega_{i,0}, \omega_{i,1}$ for each output wire i, $1 \leq i \leq \ell$. Then, it builds a table $T = ((\omega_{1,0}, \omega_{1,1}), (\omega_{2,0}, \omega_{2,1}), \ldots, (\omega_{s,0}, \omega_{\ell,1}))$ and sends it to the circuit evaluator as part of the circuit. The circuit evaluator uses T to interpret the output of the circuit: at the end of the circuit evaluation, it learns ℓ labels $w_{1,b_1}, \ldots, w_{\ell,b_\ell}$, and determines their bit value by comparing them with the values in T.

In this paper, we build on the modified garbled circuit construction of Šeděnka et al. [51]. The construction differs from "traditional" garbled circuit evaluation in three ways: (1) output labels are selected independently, even when they are computed as the output of XOR gates; (2) T is not revealed to the circuit evaluator; and (3) the circuit constructor aborts if any of the label returned by the evaluator is not in T, if not all labels are returned, or if two labels for the same bit are returned. These modifications guarantee that the resulting protocol is secure against a malicious evaluator, which is unable to alter the result of the circuit computation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

Oblivious Transfer. In order to learn the labels corresponding to its input wires, the circuit evaluator executes several instances of 1-out-of-2 oblivious transfer (OT) protocols with the circuit constructor. In 1-out-of-2 Oblivious Transfer (OT_1^2), one party (the *sender*) has two strings m_0, m_1, and the other party (the *receiver*) has one bit (b) as its input. At the end of the protocol, the receiver learns m_b while the sender learns nothing. Similarly, in 1-out-of-N OT the receiver obtains one of the N strings held by the sender.

In this paper we use an efficient implementation of OT_1^2 from [35] as well as techniques from [1] that reduce a large number of OT protocol executions to $O(\kappa)$ executions.

IV. Model and Definitions

A. Protocol Participants and Interactions

Our protocol involves three parties: the *smartphone* (owned by the user), the *Cloud*, and the *server*. The parties interact in two protocol phases: enrollment (typically executed once per user), and authentication. During enrollment, the smartphone collects and processes a user's biometric template Y. Then, it encrypts Y and sends the resulting vector $\overline{Y}$ to the server. The server has no access to the template decryption key, and is therefore unable to extract any information about the template. The Cloud does not take part in the enrollment phase.

During the authentication phase, the smartphone and the server interact with the Cloud in order to execute the protocol. The smartphone's protocol input is a biometric vector X containing biometric features, and the template decryption key. The server's input is the user's encrypted template $\overline{Y}$. The server's output is the authentication score. The smartphone has no output, and the Cloud has no input or output.

During the execution of the protocol, the server learns no information about the user's biometric template and authentication sample. Similarly, the Cloud learns no information about any of the other parties' input or output.

Our approach assumes that the smartphone has Internet connectivity, which is a common expectation of modern smartphone operating systems and apps. If the adversary attempts to circumvent the proposed architecture by interrupting network connectivity during authentication, the smartphone can fall back to offline access control policies, such as activating login-time password, or disabling access to local data. We consider offline policies to be outside the scope of this work.

B. Security Model

Many privacy-preserving protocols and protocol construction techniques are secure against semi-honest (or honest-but-curious) adversaries. Informally, semi-honest adversaries faithfully execute all protocol steps, and try to learn additional information from the transcripts of the protocol execution. This setting is appropriate when the parties can be safely assumed to behave properly, e.g., when the execution of the protocol can be audited, or when the parties have strong external incentives to not have access to any information besides the protocol output. In contrast to semi-honest parties, malicious adversaries are assumed to deviate from the intended protocol execution, and are therefore much more powerful than semi-honest adversaries.

We argue that, in the context of authentication performed on user-controlled devices, security against semi-honest adversaries is not sufficient. The adversary should, in fact, be assumed to be willing (and able) to arbitrarily deviate from the protocol if this provides any advantage when performing authentication. For example, if the adversary can successfully authenticate without knowledge of the authentication secret by sending maliciously crafted messages, then we must consider this a viable adversarial strategy. If a protocol is secure against malicious adversaries, then attacks that rely on deviating from the protocol are ineffective: security in the malicious model implies that all parties will either receive the correct output with respect to all participants' input, or receive no output.

Because our scenario involves three parties (the smartphone, the Cloud, and the server), we must assume that any two of them can collude. Next, we comment on the validity of each of the collusion scenario.

We envision that the smartphone will have some level of control over the Cloud (possibly because the Cloud is a laptop or a virtual machine owned by the user), and therefore security against colluding Cloud and smartphone is a natural requirement.

Collusion of server and smartphone is not meaningful because the Cloud has no input and no output in the protocol.

Finally, the Cloud and the server might collude. Our model does not consider this scenario. It is still an open problem whether collusion between the Cloud and the server can be addressed efficiently, and without the use of fully homomorphic encryption [22]. However, we believe that assuming non-collusion between these two parties is reasonable: our protocol imposes no restrictions on where the Cloud should be hosted, and it is therefore safe to assume that the user will choose a hosting facility that is not under the server's control.

Our model assumes that the smartphone is not compromised while it is in the hands of its legitimate user. This means that, for example, the smartphone is not running malicious software while the legitimate user is enrolling or authenticating. We model this by providing all decryption keys and signing keys stored on the smartphone to the adversary post-enrollment, and by not explicitly disclosing the user's biometric template to the adversary. This assumption is required not only by our approach, but by any authentication mechanism: if the adversary is able to run arbitrary code on the user's device during enrollment or authentication then it can, for example, capture all passwords, keystrokes, and behavioral biometric traits. This information could then be used to impersonate the user (see, e.g., [43], [40], [45]).

C. Security Definitions

We follow the security definitions of Carter et al. [8] and Kamara et al. [22], which are based on the ideal world/real world paradigm [16]. In the ideal world, there are three protocol participants, denoted as P_1, P_2, and P_3, who interact with a trusted third party (TTP). The TTP is in charge of evaluating

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

function f on the participants' input. At the beginning of the protocol execution, all parties receive their input. One party, which represents the Cloud, has no input, and receives no output from the TTP. Each protocol participant sends its inputs to the trusted third party via secure channels. The trusted third party then sends each protocol participant the respective output on the same channels.

Some subset of these parties, indicated as $\{A_i\}_{i\leq 3}$ are corrupted, and can deviate arbitrarily from the protocol. For all parties, let OUT_i be the output of P_i. The i-th partial output of an ideal protocol execution with input X is defined as:

$$\text{IDEAL}^{(i)}(\kappa, X; r) \triangleq \{OUT_j \ : \ j \in H\} \cup OUT_i$$

where H is the set of honest parties, r is all random coins of all participants, and κ is the security parameter.

In the real world, each party provides the same inputs as in the ideal world, and is additionally given access to random coins. For the i-th party, let OUT_i be its output. Then, the i-th partial output of an ideal protocol execution in the presence of $m \leq 3$ independent malicious simulators $S = (Sim_1, Sim_2, Sim_3)$ is defined as:

$$\text{REAL}^{(i)}(\kappa, X; r) \triangleq \{OUT_j \ : \ j \in H\} \cup OUT_i$$

where H, r, and κ are defined as before. Given this model, security is formally defined as:

Definition 1: An outsourced protocol securely computes the function f if there exists a set of probabilistic polynomial-time simulators $\{Sim_1, Sim_2, Sim_3\}$ such that for all probabilistic polynomial time adversaries (A_1, A_2, A_3), inputs X, auxiliary inputs, and for all $i \in \{1, 2, 3\}$:

$$\{\text{REAL}^{(i)}(\kappa, X; r)\} \stackrel{c}{\equiv} \{\text{IDEAL}^{(i)}(\kappa, X; r)\}$$

where Sim_i is the simulator for A_i, and r is uniformly random.

V. Our Approach

The goal of our protocol is to minimize the amount of computation performed by the smartphone, while also reducing the protocol execution time. This is done by outsourcing almost all the computation associated with garbled circuit evaluation to an untrusted Cloud, and by removing cut-and-choose, while guaranteeing security against malicious parties.

Our protocol does not rely on cut-and-choose because both parties with input (i.e., the smartphone and the server) are guaranteed to use a circuit that has been constructed correctly. Similarly, if either the server or the smartphone are semi-honest, then the Cloud is also guaranteed to be evaluating the correct circuit. To achieve this, we use a strategy that, although not applicable in general, is well suited for authentication. In our protocol, the smartphone acts as circuit constructor, and the Cloud as circuit evaluator. The server verifies the correctness of the circuit. During enrollment, and after each successful authentication, the smartphone constructs and signs a garbled circuit that implements the verifier. The circuit, together with its input keys, is sent to the server. The server decrypts all gates and verifies circuit correctness. If at least one of the two parties is semi-honest, the circuit to be evaluated is correct. The smartphone then deletes all information related to the circuit. Evaluation of the circuit, including oblivious transfer, is performed by the Cloud, which must prove to the smartphone and to the server that it has faithfully followed the protocol.

A malicious (or even semi-honest) smartphone can decide not to delete the circuit's input keys after sending them to the server. By retaining the input keys, the smartphone can reconstruct the server's input by comparing the server's input labels with the values generated during circuit construction. (This requires collusion between the smartphone and the Cloud, which is allowed in our model.) We argue that this is not a problem for authentication. In fact, a malicious smartphone can send a new circuit only after a successful authentication. Successful authentication requires knowledge of a *faithful* representation of the user's template. Therefore, learning the value of the template after successful authentication is arguably of no use for a malicious smartphone. Similarly, one could assume that when the smartphone is in the hands of its legitimate user, it might decide not to delete circuit information, and later use this information to authenticate without user action. However, it can as well decide to maintain a copy of the user's biometric data post-authentication. This will render any authentication protocol insecure, and therefore deletion of transient information after login is a common implicit assumption.

Because our protocol does not rely on cut-and-choose, its execution time, power consumption, and design complexity are significantly lower than with current outsourcing techniques [22], [9], [8]. This simplification allows us to offload the oblivious transfer and the circuit evaluation to the Cloud and the server, except for a small number of inexpensive checks that the smartphone performs during protocol execution.

As confirmed by our experiments, circuit construction on the smartphone is inexpensive for even relatively complex functions. Moreover, the smartphone can construct the garbled circuit offline (e.g., overnight while charging). This optimization further reduces computation and communication performed by the smartphone during protocol execution, bringing the smartphone's energy consumption to a negligible level.

A. Protocol Overview

In this section, we describe a simplified version of the enrollment and authentication phases of our protocol, and discuss the protocol design rationale. Detailed protocol description is presented in Section VI.

Enrollment Phase. During enrollment, the smartphone collects a set of biometric samples from the user, and uses them to construct the template Y. Each feature of the template is then encrypted by adding a random vector R of appropriate length to Y. R is then stored on the smartphone. Let $\overline{Y}$ be the resulting encrypted template, i.e., $\overline{Y} = Y + R$. The smartphone then constructs a set of garbled circuits that implement the authentication function, and signs them using the user's private key. Let n be the number of biometric features used for authentication, and v the bit-length of the features. In order

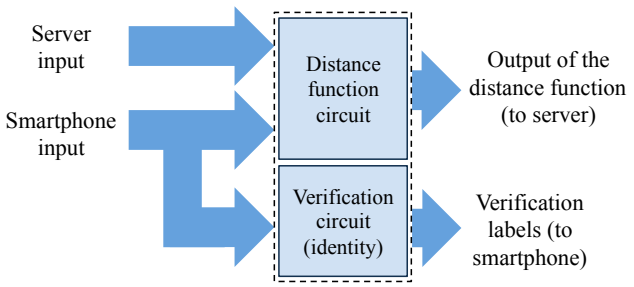

Fig. 1. High-level overview of the authentication circuit.

to later allow the smartphone to verify the correctness of the circuit's input, each circuit is augmented with $n \cdot v$ output wires, which are connected directly to the smartphone's input wires. In other words, these output wires correspond to the output of the identity function computed on the smartphone's input (indicated as "verification circuit" in Figure 1). Finally, the circuits are sent to the server, together with all the circuits' decryption keys. The server verifies the correctness of the circuits, and stores them. At the end of the enrollment process, the smartphone deletes all information, except for R and the user's private (signing) key.

Authentication Phase. The steps performed during authentication are depicted in Figure 2, and can be summarized as follows. The server sends one signed circuit to the Cloud. The Cloud verifies the signature, and aborts if the verification fails.

The smartphone collects a biometric sample, constructs an authentication vector X, and encrypts it by adding R to it. Let $\overline{X}$ be the resulting ciphertext, i.e., $\overline{X} = X + R$. The smartphone sends Z to the server, and $\overline{X} \oplus Z$ to the Cloud. Neither the Cloud nor the server can reconstruct the smartphone's input, because Z carries no information on X, and $\overline{X} \oplus Z$ is effectively a one-time pad encryption of $\overline{X}$.

The Cloud then engages in an instance of OT with the server, in which it acts as *receiver*, while the server acts as *sender*. The Cloud uses $\overline{X} \oplus Z$ as its input for the OT protocol, while the server's input is composed of the circuit's input labels. However, for each bit set to 1 in Z, the server swaps the label corresponding to 0 with the label corresponding to 1. This way, the server effectively "removes" Z from $\overline{X} \oplus Z$ via OT, and therefore the Cloud learns the input keys corresponding to $\overline{X}$.

As soon as the server sends the input labels corresponding to $\overline{Y}$ to the Cloud, the Cloud has access to all the information needed to evaluate the garbled circuit. The Cloud first computes the output labels of the verification circuit, and sends their hash to the smartphone. At the same time, the server sends all verification labels to the smartphone, together with the corresponding signature generated by the smartphone during enrollment. The smartphone computes the hash of the verification labels from the server corresponding to $\overline{X}$, and verifies that it matches with the hash received from the Cloud. The verification result is reported to both the Cloud and the server.

At this stage, the Cloud evaluates the rest of the circuit (i.e., the *Distance function circuit* in Figure 1). At the end of the evaluation, the Cloud releases the output of the distance function sub-circuit (but not the verification labels) to the server, which is now able to determine the distance between Y and X. In fact, the distance between X and Y is the same as the distance between $\overline{Y}$ and $\overline{X}$. If the distance is below a predefine threshold, the server accepts the user's identity claim and requests one or more signed circuits from the smartphone.

Design Rationale. The verification gates guarantee that neither the server nor the Cloud have tampered with the smartphone's input during the protocol. Without these gates, a malicious Cloud could undetectably flip any of the bits of $\overline{X} \oplus Z$, therefore altering one or more bits of X. Similarly, the server could flip any bit of Z, leading to the same outcome.

The same input values are used as input to the distance function sub-circuit and to the verification sub-circuit (i.e., the identity circuit). A valid signature on the circuit guarantees to the Cloud that both sub-circuits are correct, because an honest smartphone would construct a correct circuit, and an honest server would not accept an incorrect circuit. Therefore, a correct output of the verification circuits guarantees that the values of the input gates faithfully represent the smartphone's input, either because the Cloud and the server followed the protocol, or because they independently flipped the same bits in Z and $\overline{X} \oplus Z$—which also leads to a correct protocol execution.

Although it might look like the same result could be obtained by removing the identity circuit and simply comparing the circuit's input with the expected input labels, this would not lead to a secure construction. In fact, doing so requires the smartphone to keep a copy of all input labels corresponding to its input. This allows colluding Cloud and smartphone to evaluate the circuit multiple times using different input values, without any interaction with the server. This will likely lead to the disclosure of a significant portion of the server's input bits. Instead, with our approach, the smartphone does not learn any of the input gates, and learning the output of the verification gates is not useful towards the evaluation of the circuit on different inputs.

From the energy standpoint, sending information via WiFi or cellular is expensive. In order to minimize energy consumption, in our protocol the smartphone sends a small seed s (e.g., $s \leftarrow \{0,1\}^{\kappa}$) to the server, instead of the entire circuit. The seed is then used by the server to deterministically generate the garbled circuit, thus reducing communication overhead.

VI. Protocol Description

The notation used in the rest of the paper is summarized in Table I. We refer to the output labels of the identity sub-circuit, depicted in Figure 1, as "verification labels". A garbled circuit Cir_i, together with all its input and output labels, is constructed via a deterministic algorithm that takes in input the circuit's topology and a seed s_i, with $|s_i| = \kappa$. Therefore, two parties can construct identical garbled circuits, including all input and output labels, given the same circuit topology and s_i.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

TABLE I
SYMBOLS USED IN THIS PAPER

κ	Security parameter	(sk_u, pk_u)	User's signing keypair. sk_u is available only to the smartphone
Cir_i	Garbled circuit i	δ_i	Signature computed on Cir_i using the smartphone's signing key
m	Number of circuits being created for evaluation	s_i	Random seed of length κ used to select all randomness associated with Cir_i
r_i	Encryption/decryption key of i-th feature	v	Length of feature representation
$x[j]$	j-th bit of x	n	Number of biometric features for authentication
$\omega_{j,b}$	Label corresponding to bit b on wire j	T	Table matching output labels with their bit value
Y, X	User template (Y) and authentication sample (X)	T_2	Subset of T from Cir corresponding to the outputs of the verification gates
Z	Vector of random bits $(z_1, \ldots, z_{n \cdot v})$	R	Vector of random values $r_1, \ldots, r_n$, with $0 \leq r_i \leq 2^v$
$\overline{Y}$	Encrypted template ($\overline{Y} = Y + R$)	$\overline{X}$	Encrypted authentication sample ($\overline{X} = X + R$)
ℓ	Number of circuit output wires	z_i	Encryption/decryption key for the i-th bit of the authentication sample
γ	Signature on T_2		

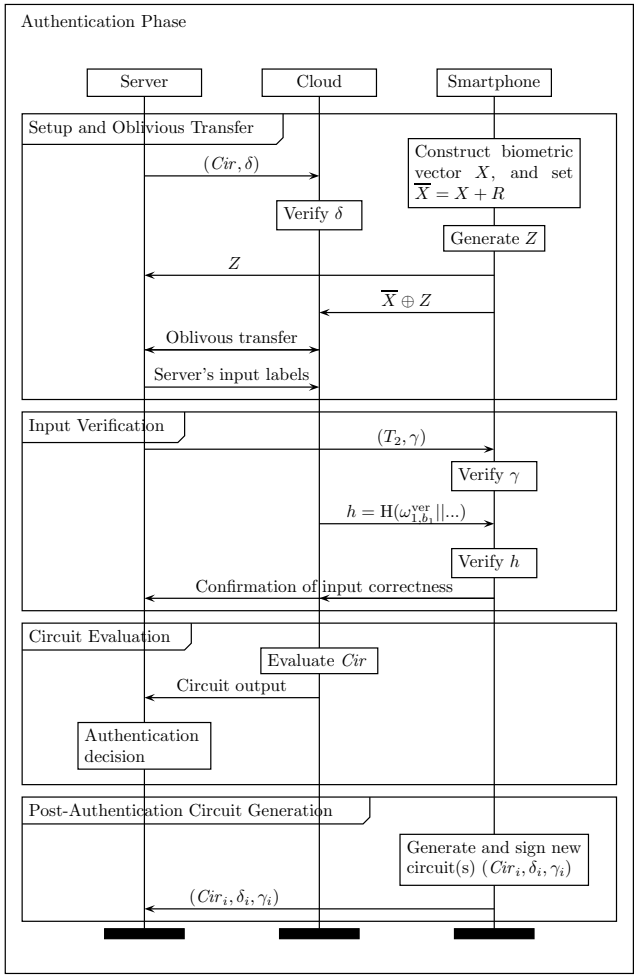

Fig. 2. Overview of our outsourced authentication protocol.

A. Enrollment Phase

Common inputs: security parameter κ, user's public key pk_u, n, v, function description $f(X, Y)$, m.

Private inputs: smartphone: Y, user's signing key sk_u; server and Cloud: none.

Outputs: server: $\overline{Y}$; smartphone and Cloud: none.

During enrollment, the smartphone selects a set of m short seeds $s_1, \ldots, s_m$. Each s_i is used to construct garbled circuit Cir_i, $1 \leq i \leq m$. Each circuit implements the same distance function (e.g., scaled Manhattan distance, or any other translation-invariant metric), and is signed using the user's signing key sk_u. In particular, the smartphone signs the garbled representation of the circuit (without including input and output keys) as $\delta_1, \ldots, \delta_m$. It also signs all tables $T_2^1, \ldots, T_2^m$ (i.e., the table which associates verification output labels with their meaning) as $\gamma_1, \ldots, \gamma_m$. Signatures δ_i-s and γ_i-s are sent to the server, together with $s_1, \ldots, s_m$. The smartphone immediately discards all the information about the circuits (i.e., it deletes all s_i-s, input keys, output wires, garbled gates values, and intermediate keys). After receiving s_i-s and δ_i-s, the server constructs $Cir_1, \ldots, Cir_m$ and verifies the signatures computed on the circuits using the user's public key pk_u. If the verification is successful, the server accepts the circuits. After constructing a feature vector $Y = (y_i, \ldots, y_n)$ representing the user's template, the smartphone selects n independent random values r_i such that $0 \leq r_i \leq 2^v$ for some v that represents the length of the features. It then sends $\overline{Y} = ((y_1 + r_1 \bmod 2^v), \ldots, (y_n + r_n \bmod 2^v))$ to the server. The smartphone stores $R = (r_1, \ldots, r_n)$ locally, and deletes Y and $\overline{Y}$.

B. Authentication Phase

Common inputs: security parameter κ, user's public key pk_u, n, v, function description $f(X, Y)$, m.

Private inputs: smartphone: X, R, user's signing key sk_u; server: $\overline{Y}$, $\{(Cir_i, \delta_i, \gamma_i)\}_{i \in [1,m]}$; Cloud: none.

Outputs: server: $f(X, Y)$; smartphone and Cloud: none.

Authentication—Phase 1: Setup and Oblivious Transfer. The server selects $(Cir, \delta, \gamma) \in \{(Cir_i, \delta_i, \gamma_i)\}_{i \in [1,m]}$, and removes it from the set of available signed circuits. Then, it sends (Cir, δ) to the Cloud. The Cloud verifies δ using the smartphone's public key. If the verification fails, the Cloud aborts protocol execution and alerts the smartphone.

The smartphone computes $\overline{X} = X + R$, selects $n \cdot v$ random bits $Z = (z_1, \ldots, z_{n \cdot v})$, and computes $\overline{X} \oplus Z$ as the bit-wise XOR of $z_1, \ldots, z_{n \cdot v}$ with $\overline{X}$. The smartphone then sends $\overline{X} \oplus Z$ to the Cloud, and Z to the server. Because the Cloud does not know any of the z_i, it cannot extract any information from $\overline{X}$. Similarly, in this stage the server learns $n \cdot v$ random bits (i.e., Z), which reveal no information about the smartphone's input.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

The Cloud and the server engage in the OT protocol, where the server acts as sender and the Cloud acts as receiver. The Cloud's input is $\overline{X} \oplus Z$, while the server's is the circuit's input labels. The server's input, however, is modified as follows. For each OT_1^2, the inputs of the server are "swapped" if the corresponding z_i is 1. Let $\omega_{i,0}$ and $\omega_{i,1}$ be the labels for the i-th input wire of the garbled circuit. If $z_i = 1$, then the values of $\omega_{i,0}$ and $\omega_{i,1}$ are swapped. This way, at the end of the execution of the OT protocol, the Cloud learns the input labels corresponding to $\overline{X}$, rather than to $\overline{X} \oplus Z$. However, because the labels are computed independently from their corresponding bit value, the Cloud does not learn the smartphone's input. Finally, the server sends the input labels corresponding to $\overline{Y}$ to the Cloud.

Authentication—Phase 2: Input Verification. The Cloud uses the circuit input labels obtained in the previous phase to evaluate the verification sub-circuit. This amounts to decrypting the verification sub-circuit output gates using the input labels as decryption keys. The output labels are then deterministically encoded as a single string (e.g., via concatenation as $\omega_{1,b_1} || \ldots || \omega_{(n \cdot v),b_{(n \cdot v)}}$), which is hashed and sent to the smartphone.

The server sends the table T_2 corresponding to Cir to the smartphone, together with γ. Let $T_2 = ((\omega_{1,0}^{\text{ver}}, \omega_{1,1}^{\text{ver}}), \ldots, (\omega_{(n \cdot v),0}^{\text{ver}}, \omega_{(n \cdot v),1}^{\text{ver}}))$. The smartphone verifies γ, then computes $\text{H}(\omega_{1,b_1}^{\text{ver}} || \ldots || \omega_{(n \cdot v),b_{(n \cdot v)}}^{\text{ver}})$ where b_i is the i-th bit of $\overline{X}$, and compares it to the hash received from the Cloud. If the two values are not equal, then the smartphone aborts protocol execution. Otherwise, it notifies the Cloud and the server that the verification completed successfully.

Authentication—Phase 3: Circuit Evaluation. The Cloud evaluates the distance function sub-circuit of Cir (see Figure 1), then sends the corresponding output labels (and withholds the verification output labels) to the server. The server checks that the values received from the Cloud are correct by verifying that: (1) the Cloud returned exactly one value per output wire; and (2) each value corresponds to one of the output wire values. Then, the server interprets the output labels to reconstruct the protocol output. The server makes an authentication decision based on this output.

Authentication—Phase 4: Post-Authentication Circuit Generation. If the authentication is successful, the server asks the smartphone to send one more signed garbled circuit. Otherwise, the server rejects the smartphone, and possibly repeats the authentication process with another circuit that has not been evaluated before.

C. Protocol Correctness

In this section, we show that if all parties faithfully follow all protocol steps, the protocol outputs the correct result. Correctness of the protocol in presence of malicious adversaries is addressed in Section VII.

Enrollment Phase. During enrollment, the smartphone generates m circuits using seeds $s_1, \ldots, s_m$, and signs them and the corresponding tables $T_2^1, \ldots, T_2^m$. Because generating garbled circuit representations and the tables from each seed is a deterministic process, the server generates the same circuit representations as the smartphone. Therefore signatures $\delta_1, \ldots, \delta_m$ and $\gamma_1, \ldots, \gamma_m$ are valid.

Authentication Phase. During Setup and Oblivious Transfer (Authentication—Phase 1), the Cloud uses $\overline{X} \oplus Z$ as its input to the OT protocol. The server's input consists in the circuit labels; for each $z_i = 1$, the server swaps the corresponding input labels as detailed in Section VI-B. In particular, for each $z_i = 0$, label ω_{i,b_i} corresponds to input bit b_i, and for each $z_i = 1$, label ω_{i,b_i} corresponds to input bit $1 - b_i$. In other words, the label associated by the server to bit b_i is $\omega_{i,(b_i \oplus z_i)}$. For each $z_i = 0$, the Cloud requests ω_{i,b_i}-s corresponding to $\overline{X} \oplus Z$ to the server via OT. The i-th bit of $\overline{X} \oplus Z$ is the XOR of z_i with bit b_i of $\overline{X}$, and therefore, the Cloud effectively requests labels corresponding to $b_i \oplus z_i$ to the server via OT. Hence, the Cloud receives labels $\omega_{i,(b_i \oplus z_i) \oplus z_i} = \omega_{i,b_i}$, and at the end of the protocol the Cloud learns the input labels corresponding to $\overline{X}$.

During the input verification phase (Authentication—Phase 2), the hash value received by the smartphone from the Cloud is equal to the hash of the verification sub-circuit's output labels corresponding to $\overline{X}$, because (1) the Cloud obtained the correct input labels from the server via OT; (2) the garbled circuit is constructed correctly; and (3) the smartphone and the Cloud use the same (deterministic) encoding and hash function.

The verification circuit computes a translation-invariant distance (e.g., Manhattan or Euclidean distance) between $\overline{X} = X + R$ and $\overline{Y} = Y + R$, which is the same as the distance between X and Y. For example, with Manhattan distance, $f(\overline{X}, \overline{Y}) = \sum_{i=1}^{n} |(x_i + r_i) - (y_i + r_i)| = \sum_{i=1}^{n} |x_i - y_i| = f(X, Y)$. Therefore, the protocol computes the correct distance between the template and the biometric sample.

D. Smartphone Output

With the protocol presented in this section, the server is the only party with output. The protocol can be modified to provide independent outputs to the smartphone and the server as follows. Let O_A be the subset of the circuit's output wires that correspond to the server's output, and O_B the subset corresponding to the smartphone's output. The server reveals the portion of T corresponding to O_B (denoted as T_B) to the smartphone, together with the signature on T_B computed by the smartphone during circuit construction, and withholds the part corresponding to $O_A \setminus O_B$. At the end of the circuit evaluation, the Cloud reveals O_A to the server and O_B to the smartphone.

E. Cost of Our Approach

Server. For each protocol execution, the server verifies one circuit received from the smartphone, which entails performing three encryptions for each non-XOR gate and verifying two signatures, computed on the entire circuit and on the verification output labels. The server also acts as *sender* in the instance

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

run with the Cloud, where it encrypts the receiver's OT input using one-time pad with the key provided by the smartphone. Therefore, the cost for the server is $O(n \cdot v) + O(|Cir|)$, where $|Cir|$ indicates the size of the garbled circuit. In terms of communication, the server's cost is $n \cdot v$ bits from the smartphone (corresponding to the OT decryption keys), $O(\kappa^2) + O(n \cdot v)$ (OT with the Cloud), $O(|Cir|)$ (sending the circuit to the Cloud) and $O(|f(X,Y)|)$ (receiving the output of the protocol from the Cloud).

Smartphone. The smartphone constructs a garbled circuit and signs it, which entails three encryptions for each non-XOR gate and two instances of signature generation. During the authentication phase, the smartphone computes a one-time pad encryption of its input, and verifies that the result of the input verification from the Cloud is correct (this requires one invocation of a hash function). The communication cost on the smartphone is $n \cdot v$ bits sent to the server, $O(|T_2|) + O(\kappa)$ from the server, $n \cdot v$ bits sent to the Cloud, and $O(\kappa)$ to the server post-authentication.

Cloud. During authentication, the Cloud acts as receiver in the OT protocol executed with the server. It then evaluates the circuit. The cost for the Cloud is therefore $O(n \cdot v) + O(|Cir|) + O(\kappa)$. The communication cost for the Cloud is $n \cdot v$ bits from the smartphone for the OT input, $O(\kappa^2) + O(n \cdot v)$ for performing OT with the server, $O(|Cir|)$ for receiving a circuit from the server, $O(n \cdot v)$ for sending the verification gates to the smartphone, and $O(|f(X,Y)|)$ for sending the output of the computation to the server.

In terms of storage, the server has a copy of the encrypted biometric template, one or more circuits from the smartphone, and the user's public key. The smartphone stores the template decryption key and the signing keypair. The Cloud needs to store only the user's public key in order to verify the signature on the circuits.

F. Evaluating Arbitrary Functionalities

If we assume that the Cloud and the smartphone do not collude (as in Kamara et al. [22] and Carter et al. [9]), our protocol is suitable for evaluating arbitrary functionalities, and is therefore not limited to authentication. However, we believe that assuming non-collusion is unrealistic in most scenarios. It is still an open problem how to evaluate arbitrary functions in presence of malicious and colluding Cloud and smartphone, without using cut-and-choose.

VII. Security Proofs

In this section, we show that our approach is secure: (1) when then Cloud is malicious, and the smartphone and the server are semi-honest (Theorem 1); (2) when the Cloud and the server are semi-honest, and the smartphone is malicious (Theorem 2); (3) when the Cloud and the smartphone are semi-honest, and the server is malicious (Theorem 3); and (4) when the smartphone and the Cloud are malicious and colluding (Theorem 4).

Throughout this section, we assume that the OT protocol used in our construction is secure in the malicious model, and therefore simulatable. Further, we assume that the garbled circuit construction of Šeděnka [51] is secure against malicious clients and semi-honest servers. Finally, the signature scheme used in our protocol is existentially unforgeable, and hash function $\mathrm{H}(\cdot)$ is modeled as a random oracle.

A. Security Against a Malicious Cloud

We start by showing that a malicious Cloud cannot learn any information about the inputs or outputs of the protocol besides their size (privacy), nor it can affect the result of the computation except with negligible probability (correctness).

To prove that our protocol provides privacy to the smartphone and the server, we need to show that the Cloud either cannot distinguish correct protocol messages from random values, or that correctly distributed messages can be constructed without knowledge of the inputs. Informally, this is true because of the following reasons. During authentication, the Cloud receives: (1) Cir and δ—because Cir is constructed independently from the parties' private inputs, it reveals no information on them; (2) $\overline{X} \oplus Z$ corresponds to the one-time-pad encryption of $\overline{X}$, and therefore appears to be random and uniformly distributed to the Cloud; (3) the messages exchanged during the OT protocol—because the underlying OT protocol is secure in the malicious model, the Cloud learns nothing from the execution besides the protocol's outputs; and (4) the outputs of the OT protocol, which is a set of random strings corresponding to the circuit input labels.

To show that the protocol enforces correctness, we prove that any deviation from the intended protocol steps which affects the protocol output can always be identified by the smartphone and/or the server. Clearly, the Cloud can abort at any time. However, both the server and the smartphone will notice it. If the Cloud does not abort, it can deviate from the correct execution of the protocol as follows: (1) it can use $S \neq \overline{X} \oplus Z$ as input to the OT protocol—this adversarial strategy is identified by the smartphone by checking the outputs of the verification sub-circuit; (2) it can alter the hash value sent to the smartphone—however, this strategy can be trivially identified by the smartphone, because the Cloud cannot forge output labels corresponding to inputs to the OT protocol other than the one used; and (3) it can alter the output labels sent to the server—this is identified by the server because the Cloud cannot reconstruct any label that does not correspond to the circuit's correct output, and the server knows the list of all output labels. (Because the OT protocol is secure against malicious adversaries, no efficient adversarial strategy can cause the OT protocol to output the incorrect result.)

Next, we formalize this intuitive security argument.

Theorem 1: The protocol in Section VI securely computes a function f according to Definition 1 when the Cloud is malicious, while the smartphone and the server are semi-honest.

Proof: To prove Theorem 1, we show that it is possible to construct two simulators, Sim_1 and Sim_2, that act as the simulator for the smartphone and the simulator for the server, respectively (the two simulators share their internal state).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

The two simulators interact with the malicious Cloud (the adversary), and act as follows.

During enrollment: The Cloud does not exchange any message with the other parties during enrollment, and therefore the simulators do not interact with the adversary in this stage.

During authentication, Phase 1-4: Sim_1 selects a uniform random value V from $\{1,\dots,2^v\}^n$ and sends it to the adversary. From the adversary's point of view, V follows the same distribution as $\overline{X} \oplus Z$: in our protocol, Z is uniformly distributed in $\{1,\dots,2^v\}^n$, as is therefore $\overline{X} \oplus Z$. Sim_1 then selects a random seed s_1, and uses it to generate circuit Cir according to the protocol. Then Sim_1 signs Cir, generating signature δ_1. Sim_2 then sends (Cir, δ) to the adversary.

Sim_2 and the adversary engage in OT. Sim_2 acts as OT simulator for the underlying OT protocol, using input labels from Cir. (Sim_2 successfully acts as OT simulator iff the underlying OT protocol is secure against malicious adversaries.) At the end of the protocol, Sim_2 obtains the adversary's input S. Sim_2 then sends Cir to the adversary, together with a subset of the server's input labels corresponding to a uniformly distributed random input. The labels are generated independently from the parties' inputs, and therefore the adversary cannot tell the input labels it obtains are a related to the protocol inputs.

Eventually, the adversary sends h_{Adv} to Sim_1. Sim_1 computes $h = \mathrm{H}(\omega^{\mathrm{ver}}_{1,b_1} || \dots || \omega^{\mathrm{ver}}_{(n\cdot v),b_{(n\cdot v)}})$, where b_i is the i-th bit of V. If $h_{Adv} \neq h$, then Sim_1 aborts. Because of the security of the underlying OT and garbled circuit construction, and because $\mathrm{H}(\cdot)$ is a random oracle, the adversary can send $h_{Adv} = h$ iff its input to the OT protocol is $S = V$. In fact, let b_j be the first bit of S that differs from the corresponding bit in V. At the end of the OT, the adversary learns the input label corresponding to b_j, which does allows it to decrypt the verification output label corresponding to $1 - b_j$, which is needed to calculate h.

The adversary then sends the output of the circuit evaluation to Sim_2, which checks output correctness against T, and aborts if the adversary's output is incorrect.

Therefore, Sim_1 and Sim_2 can be constructed in such a way that the adversary cannot distinguish them from honest protocol participants. This proves Theorem 1. □

B. Security Against a Malicious Smartphone

To prove that a malicious smartphone cannot learn any information on the template, nor can cause the protocol to output an authentication score that is different from the score correctly computed from its inputs, we need to show that all messages received by the smartphone are either indistinguishable from random, or can be computed without knowledge of the protocol inputs. Further, we need to show that deviating from the protocol does not give any advantage to the adversary. We address this by showing that, for each protocol deviation, there is an equivalent protocol input modification that the smartphone can compute with no knowledge of the template.

Informally, all messages received by the smartphone during the protocol reveal no information on the template because: (1) T_2 and γ are generated by the smartphone during the enrollment phase independently from the user's template; and (2) hash h from the Cloud is computed on a subset of the elements of T_2 selected by the adversary using $\overline{X}$ and Z.

The smartphone can alter the following messages: (1) Z, sent to the server, and $\overline{X} \oplus Z$, sent to the Cloud—the two modifications are equivalent, and for any two strings S_1, S_2 used to replace Z and $\overline{X} \oplus Z$, the adversary could have computed the equivalent X as $(S_1 \oplus S_2) - R$; (2) the confirmation of input correctness, sent to the Cloud and the server—altering this value will causes the protocol to abort for either the Cloud, the server, or both, and does not otherwise affect the protocol outputs.

This informal security argument is formalized next. In the proof, the simulators run the protocol in the ideal world with the TTP, and in the real world with the adversary [16].

Theorem 2: The authentication phase of the protocol in Section VI securely computes a function f according to Definition 1 when the smartphone is malicious, and both the server and the Cloud are semi-honest.

Proof: We prove Theorem 2 by constructing two algorithms, henceforth Sim_1 and Sim_2, which act as the simulator for the Cloud and the simulator for the server, respectively (the two simulators share their internal state). Sim_1 and Sim_2 interact with the malicious smartphone (the adversary), and extract the smartphone's input as follows.

During the enrollment: Because the smartphone is not compromised during enrollment (see Section IV-B), Sim_2 receives the user's encrypted template $\overline{Y}$, seeds $s_1,\dots,s_m$, and signatures $\delta_1,\dots,\delta_m$ and $\gamma_1,\dots,\gamma_m$. If the signatures are correct, then Sim_2 forwards $\overline{Y}$ to the TTP.

During authentication, Phases 1-4: The adversary receives R and sk_u. Then, it sends Z to Sim_2, and $\overline{X}^*$ to Sim_1. Sim_2 recovers the adversary's input as $\overline{X}^* \oplus Z$, and sends it to the TTP. In the remainder of the protocol, Sim_1 and Sim_2 faithfully follow all protocol steps. In particular, the adversary's view includes (T_2, γ), which is independent from the party's inputs, and h, which corresponds to the subset of T_2 identified by $\overline{X}^* \oplus Z$. Because all messages received by the adversary during the protocol are computed consistently with the adversary's input, the adversary cannot distinguish between interacting with the simulators, and interacting with the honest Cloud and server. Therefore, the simulation is undetectable, and this proves Theorem 2. □

C. Security Against a Malicious Server

Because the smartphone and the Cloud have no output, to prove that our protocol is secure against a malicious server we need to show that the adversary cannot distinguish protocol messages from random values, or that protocol messages can be generated without knowledge of the smartphone's input. Further, we need to show that a simulator interacting with the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

server can extract the server's protocol input, and use in in the ideal world with the TTP.

Theorem 3: The protocol in Section VI securely computes a function f according to Definition 1 when the server is malicious, and both the smartphone and the Cloud are semi-honest.

Proof: In order to prove Theorem 3, we show that it is possible to construct two algorithms, Sim_1 and Sim_2, which act as the simulator for the smartphone and for the Cloud, respectively, and extract the adversary's inputs. The two simulators act as follows.

During the enrollment: In the ideal world, the server has no input, and Sim_1 receives an encrypted template $\overline{Y}^{Ideal}$ from the TTP.

Sim_1 executes the same protocols steps as the smartphone, except that $\overline{Y} = \overline{Y}^{Ideal}$. Sim_1 generates (sk_u^*, pk_u^*), faithfully constructs $Cir_1, \ldots, Cir_m$ from random seeds $s_1, \ldots, s_m$, and signs each circuit and circuit verification gates as $\delta_1, \ldots, \delta_m$ and $\gamma_1, \ldots, \gamma_m$ respectively. Then, it sends pk_u^*, $s_1, \ldots, s_m$, $\delta_1, \ldots, \delta_m$, $\gamma_1, \ldots, \gamma_m$, and $\overline{Y}$ to the server. The adversary cannot distinguish Sim_1 from the smartphone because pk_u^*, $s_1, \ldots, s_m$, $\delta_1, \ldots, \delta_m$, $\gamma_1, \ldots, \gamma_m$, and $\overline{Y}$ all follow the same distribution as the corresponding values from the smartphone.

During authentication, Phase 1: Sim_1 selects $n \cdot v$ random bits $z_1, \ldots, z_{n \cdot v}$ and sends them to the adversary.

Sim_2 engages in the OT with the adversary. The input of Sim_2 to the OT protocol is $n \cdot v$ uniformly random bits $w_1, \ldots, w_{n \cdot v}$. Because of the security of the underlying OT protocol, the adversary cannot distinguish between Sim_2 and the Cloud. Then, Sim_2 receives the adversary's input labels.

During authentication, Phase 2: The adversary sends Cir, δ, T_2, and γ to Sim_2. If the signatures do not verify, Sim_2 aborts. Also, if the signature on Cir is valid, but Cir is not among the circuits generated in the enrollment phase by Sim_1, then Sim_2 aborts. The latter abort happens with negligible probability, since the adversary is not able to return a valid signature on a circuit that was not previously signed by Sim_1.

Sim_2 "decodes" the adversary's input labels to their corresponding input bits as $\overline{Y}^*$, using all input gates of Cir (which can be reconstructed from the corresponding seed, generated during enrollment). Sim_2 then verifies that the input labels received from the adversary via OT correspond to the subset of the smartphone's input label from T_2 selected by bits $(z_1 \oplus w_1), \ldots, (z_{n \cdot v} \oplus w_{n \cdot v})$, and aborts if they do not. Sim_2 sends $\overline{Y}^*$ to the TTP.

During authentication, Phase 3: Sim_2 receives $d = f(X, Y)$ from the TTP. It encodes d using the the appropriate output labels form Cir, and sends the resulting labels to the adversary. Because $f(\overline{X}, \overline{Y}) = f(X, Y)$, the labels corresponding to d are distributed as expected by the adversary.

During authentication, Phase 4: If the value corresponds to a correct authentication decision, then Sim_1 generates a new circuit, signs it, and sends the seed and the signature to the adversary. The messages corresponding to the circuit and its signature are distributed as expected by the adversary.

This proves Theorem 3. □

D. Security Against Colluding Smartphone and Cloud

In this section, we show that even if the smartphone and the Cloud are simultaneously *malicious* and *colluding*, they still cannot learn any information on the server's input, nor trick the server to authenticate correctly without knowing the user's biometric template.

To model collusion, in this scenario the smartphone and the Cloud act as a single malicious adversary. Informally, when the smartphone and the Cloud act as a single entity, our protocol falls back to the protocol of Šeděnka et al. [51]. Because the protocol in [51] is secure against malicious clients, our protocol is secure against malicious and colluding smartphone and Cloud. In particular, we need to show that it is possible to build a simulator that successfully extracts the adversary's input, and that all messages from the simulator are indistinguishable from correct protocol messages.

Theorem 4: The protocol in Section VI securely computes a function $f(X, Y)$ according to Definition 1 when the Cloud and the smartphone are malicious and colluding, and the server is semi-honest.

Proof: The security of our protocol when the Cloud and the smartphone collude falls back to the security of the underlying garbled circuit protocol. We prove Theorem 4 by constructing simulator Sim_1, which simulates the server. Sim_1 interacts with the malicious adversary, which controls the smartphone and the Cloud, and act as follows.

During the enrollment: Because the smartphone is not compromised during enrollment, Sim_1 receives an encrypted template $\overline{Y}$, seeds $s_1, \ldots, s_m$, and signatures $\delta_1, \ldots, \delta_m$ and $\gamma_1, \ldots, \gamma_m$ from the legitimate user, while the adversary receives R and sk_u. Sim_1 verifies the correctness of the circuits and the signatures, and aborts if verification fails. This guarantees that the circuits compute the correct functionality.

During authentication, Phases 1-4: Sim_1 sends (Cir, δ) to the adversary, where Cir is constructed from one of the seeds obtained during enrollment and δ is the corresponding signature. Therefore, these two values are correctly distributed. It then receives Z from the adversary (acting as the smartphone), and interacts with the adversary acting as Cloud in the OT protocol. Because the OT protocol is secure in the malicious model, and therefore simulatable, Sim_1 uses it to extract the adversary's input $\overline{X}^*$. Then, Sim_1 sends $\overline{X}^* \oplus Z$ to the TTP, and a random subset of its input labels to the adversary. The labels are generated independently from the parties' inputs, and therefore the adversary cannot tell whether they are the correct labels for $\overline{Y}$.

Because there is no further protocol message sent from the server to the smartphone or to the Cloud, Sim_1 simply verifies that the adversary confirms the correctness of the input (Phase 2), and that it then returns a valid instance of circuit output (Phase 3).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

The adversary cannot distinguish between interacting with the simulator, and interacting with an honest server. Therefore, this proves Theorem 4. □

VIII. Comparison with Other Cloud-Aided Circuit Evaluation Protocols

In this section, we compare our approach in terms of complexity and security properties to three Cloud-aided garbled circuit evaluation techniques: Salus [22], Carter et al. [9] (CMTB hereafter), and Whitewash [8].

There are two aspects in which our protocol differs from all three Cloud-aided techniques: (1) our approach does not rely on cut-and-choose. This leads to a dramatic reduction in execution time compared to current protocols (between 30 and 130 times, depending on the protocol, functionality, and input size) and bandwidth usage (50 to 90 times); and (2) to achieve this performance efficiency, our technique sacrifices generality. In fact, if we allow collusion between the Cloud and the smartphone, our approach is suitable only for authentication.

Comparison of the asymptotic complexity of outsourcing protocols is presented in Table II. Next, we provide a brief overview of specific differences between our approach and Salus, CMTB, and Whitewash.

Comparison to Salus. Salus guarantees security only in presence of non-colluding adversaries. This is a substantially weaker model than the one addressed in our work, because we assume that the Cloud and the smartphone can collude. This limitation makes Salus unsuitable for outsourcing authentication protocols, where the smartphone owner is also the owner of the Cloud instance. With Salus [22], the smartphone generates circuit randomness and garbles its own inputs. In contrast, in our approach the smartphone also generates the circuit. Salus can implement functions which compute a single shared output value. Our protocol supports functions that compute different outputs for each protocol participant. Salus requires the use of a fair coin-flipping algorithm, executed by all participants. This is not needed in our protocol.

Comparison to CMTB. As in Salus, and in contrast to our approach, CMTB assumes that none of the parties collude with the Cloud, therefore making the protocol unsuitable for authentication. In CMTB and our approach, OT and input verification are outsourced to the Cloud. However, in CMTB, part of the OT protocol is executed by the smartphone, which therefore incurs substantial computation overhead. Finally, CMTB requires the smartphone to run a two-party fair coin toss protocol, which is not required in our protocol.

Comparison to Whitewash. Whitewash is the closest protocol to ours in terms of security model and cost for the smartphone. Both our protocol and Whitewash are secure malicious adversaries, and against colluding malicious Cloud and smartphone.

While in our approach the smartphone constructs the circuit, circuit generation is offloaded to the Cloud and the server in Whitewash. Depending on the authentication distance function, our protocol requires the smartphone to perform significantly more symmetric operations than with Whitewash. In fact, while with Whitewash the smartphone performs a number of operations that depend only on the size of its own input, with our approach the smartphone constructs the entire circuit. However, because our protocol does not rely on cut-and-choose, the cost of constructing a single circuit is, in practice, very small (see results in Section IX-C). This allows our protocol to have a better time/energy tradeoff than Whitewash. For example, our protocol computes Hamming distance with 1,600-bit input in 3.29s, compared to the computation time of 95.57s of Whitewash. In this setting, the smartphone energy cost of our protocol is 1.23 mWh, which corresponds to 0.01% of the battery of a Samsung Galaxy S4.

As with Whitewash, our approach can evaluate functionalities where the smartphone and the server obtain different outputs.

IX. Evaluation

We conducted a detailed performance and energy characterization of our technique using a commodity smartphone. We compared our approach with traditional garbled circuits (i.e., non-Cloud-aided, with and without cut-and-choose), and with the performance results reported by Carter et al. [8]. Our evaluation is performed using two circuits: the first implements scaled Manhattan distance, while the second computes Hamming distance. The former was chosen because previous work on behavioral authentication has shown that scaled Manhattan is among the top performers (see, e.g., [23], [51]). The latter, because it is used in iris matching [6], and it is a standard benchmark for garbled circuit implementations [8].

A. Experiment Setting

We used a Samsung Galaxy S4 smartphone running Android 4.3. The smartphone's CPU is a 4-Core 1.9GHz Qualcomm Snapdragon, and is combined with 2GB RAM. The battery capacity is 9,880 mWh. To measure the phone's power consumption, we connected it to a Monsoon power monitor [33], which acted as a power supply and as a data acquisition device. To obtain accurate measurements, we bypassed the battery, and powered the smartphone solely via the power monitor. To provide a realistic assessment of the cost of the various protocols, we report energy consumption for the entire system, including the screen (which was set at medium brightness), WiFi, and standard OS background processes. To evaluate the cost of the protocols on the server and the Cloud, we deployed two Dell PowerEdge R320 rack servers with Intel Xeon E5-2430L v2 6-Core 2.4GHz CPU and 64GB RAM. The servers were running Ubuntu Linux 14.04 LTS with kernel 3.16.

We performed experiments in a controlled network environment. The smartphone, the server, and the Cloud were connected to the same broadcast domain via a single access-point/switch. We used an Apple Airport Extreme Base Station, which has three Gigabit Ethernet LAN ports and 802.11ac WiFi. The smartphone was connected to the access point network through WiFi. The server and Cloud were connected to the switch through Gigabit Ethernet ports. Because the amount of data exchanged by the smartphone with our protocol is very small (between 5 KB and 780 KB), especially compared to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

TABLE II
OPERATIONS PERFORMED BY THE PARTIES. c REPRESENTS THE NUMBER OF CIRCUITS GENERATED FOR CUT-AND-CHOOSE.

	Smartphone				**Server/Cloud**
Protocol	Symmetric ops.	Asymmetric/group ops.	OT-s	Fair coin toss	Circuits evaluated
CMTB [9]	$\|X\|$	$2/5c \cdot (\|Y\|+1)$	κ	yes	$2/5c$
Salus [22]	$2/5c \cdot (\|X\|+\|Y\|+\|f(x,y)\|)$	–	0 (performed by server/Cloud)	yes	$2/5c$
Whitewash [8]	$c \cdot (\|X\|+2/5 \cdot \|f_b(X,Y)\|)$	–	0 (performed by server/Cloud)	no	$2/5c$
This work	$\|Cir\|+\|X\|$	2	0 (performed by server/Cloud)	no	1

TABLE III
INPUT SIZE AND CIRCUIT SIZE USED IN OUR EVALUATION. NUMBERS REPORTED IN BRACKETS INCLUDE VERIFICATION GATES.

	Input Size (each party, bits)	Gates	Non-XOR Gates
Manhattan distance	96 (8·12)	2,026 [2,122]	955 [1,051]
	336 (28·12)	7,516 [7,852]	3,545 [3,881]
Hamming distance	1,600	25,215 [26,815]	15,589 [17,189]
	16,384	212,912 [229,296]	114,669 [131,053]

the current state of the art (see Section IX-C), we expect that performing communication over cellular network would not meaningfully affect our results.

B. Implementation Details

The OT protocol used for the two non-Cloud-aided garbled circuit implementations (i.e., semi-honest and malicious) is from [47]. Our protocol uses the OT protocol from [1]. This led to a small reduction in terms of communication cost for our protocol (in the order of 600KB to a few MB, depending on the protocol and input size), compared to the semi-honest construction.

On the smartphone, we developed an Android application that implements our protocol, as well as semi-honest garbled circuits, and malicious cut-and-choose circuits based on the code of Kreuter et al. [26]. Code in [26] uses pipelining to reduce protocol execution time. Our implementation aims to offer 80-bit (equivalent) security ($\kappa = 80$). Therefore, the number of circuits used for cut-and-choose is $c = 256$.[1] Communication and authentication code on the smartphone were implemented in C++ as an Android NDK library. The code on the server and the Cloud was written in C++.

The scaled Manhattan distance circuit was instantiated with 8 and 28 features, represented using 12 bits ($v = 12$). These values have been shown to lead to the lowest authentication errors in [51] and in [50]. For Hamming distance, we used 1600- and 16384-bit inputs to be able to compare with [8]. We generated all signatures using the `RSA_sign()` function of OpenSSL 1.0.1, with 1024-bit RSA keys and SHA-256. We used SHA-256 to combine the output of the verification gates sent by the Cloud to the smartphone. Number of gates and approximate circuit size is presented in Table III. Comparisons between techniques presented in this paper and [8] are conservative, because the number of gates used in our Hamming distance circuit can be further reduced [7]. Relative performance improvements of our technique reported in Table IV are largely independent of circuit design.

We divide the smartphone's portion of the protocol into *online* and *offline* computation. In our implementation, circuit construction and signing is performed offline, while the rest of the protocol (including all communication) is executed online.

We did not use multi-threading to implement our technique. Likewise, the code for the two non-Cloud-based garbled circuit protocols was implemented as a single thread. This is in contrast with the implementation of Whitewash, which used MPI to divide protocol load across 64 CPU cores [8].

C. Results

Our results are summarized in tables IV and V. We report the cost of all steps within each protocol in terms of computation, communication, and energy consumption. For communication and computation, we report both the overall protocol time and the portion performed by the smartphone. We also report how many authentication attempts can be performed with each protocol before the smartphone runs out of battery, assuming that the phone is not running any other program.

The performance figures for Whitewash and CMTB listed in Table V were obtained by Carter et al. [8] using two servers with dual four-core eight-thread Intel Xeon E5620, and a Samsung Galaxy Note II smartphone with a 1.6GHz CPU. Our servers are about 1% faster than those used in [8] in single-threaded applications.[2]

As the results in tables IV and V indicate, our protocol is substantially faster than current approaches secure in the malicious model. Even though our approach is the only one in which the smartphone constructs the circuit, the resulting protocol is still considerably faster because our technique does not require cut-and-choose.

Our protocol is also faster than the garbled circuit protocol implementation secure in the semi-honest model. At first, this might seem counter-intuitive, because our approach involves more computation than its semi-honest counterpart. However, the reason we were able to achieve this result is that OT, which accounts for a substantial portion of the protocol cost, is entirely offloaded to the Cloud and the server, which are equipped with faster CPUs than the smartphone.

As reported in Table V, our approach is significantly faster than both Whitewash and CMTB. Because of its performance advantage, our protocol is presently the only one suitable for low-latency authentication windows (i.e., our protocol is the

[1] Although there are techniques that require a smaller number of circuits during cut-and-choose (e.g., $c = 40$ in [27]), the speedup resulting from reducing c does not affect the conclusions presented in this section.

[2] Based on the results from https://www.cpubenchmark.net/singleThread.html

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585093, IEEE Transactions on Information Forensics and Security

TABLE IV
EXPERIMENT RESULTS FOR SCALED MANHATTAN AND HAMMING DISTANCE. THE "# OF PROTOCOL RUNS" COLUMN INDICATES HOW MANY AUTHENTICATION ATTEMPTS CAN BE PERFORMED BEFORE EXHAUSTING A FULLY CHARGED 9,880 mWh BATTERY, WITHOUT OFFLINE PRE-COMPUTATION.

Protocol	Time on Smartphone (s)	Total Time (s)	Bandwidth on Smartphone (MB)	Total Bandwidth (MB)	Energy on Smartphone (mWh)	# of Protocol Runs
Scaled Manhattan Distance, 8 features:						
GC (semi-honest)	0.61	0.72	0.04	0.04	0.17	>56k
GC (malicious)	110.05	114.72	6.16	6.16	80.04	123
This work (malicious)	**0.04 (online) + 0.18 (offline)**	**0.35**	**0.005**	**0.06**	**<0.01 (online) + 0.05 (offline)**	**>1.7M**
Scaled Manhattan Distance, 28 features:						
GC (semi-honest)	1.80	2.00	0.15	0.15	0.76	1,300
GC (malicious)	378.69	393.29	21.78	21.78	274.87	35
This work (malicious)	**0.18 (online) + 0.42 (offline)**	**0.93**	**0.02**	**0.14**	**0.05 (online) + 0.15 (offline)**	**>48k**
Hamming Distance, 1,600 bits:						
GC (semi-honest)	7.38	9.92	0.64	0.64	5.51	1,793
GC (malicious)	1,701.30	1,753.90	95.86	95.86	1,299.19	7
This work (malicious)	**1.13 (online) + 1.15 (offline)**	**3.29**	**0.08**	**0.49**	**0.64 (online) + 0.59 (offline)**	**7,943**
Hamming Distance, 16,384 bits:						
GC (semi-honest)	70.89	105.31	6.16	6.16	64.77	152
GC (malicious)	18,125.81	19,653.31	937.66	937.66	14,758.19	0
This work (malicious)	**10.22 (online) + 9.39 (offline)**	**24.97**	**0.78**	**4.24**	**6.86 (online) + 5.67 (offline)**	**788**

TABLE V
COMPARISON WITH CURRENT CLOUD-AIDED PROTOCOLS ON HAMMING DISTANCE WITH 1,600 BIT AND 16,384 BIT INPUT.

	1,600-bit Input		16,384-bit Input	
Circuit	Time (s)	Bandwitdh (MB)	Time (s)	Bandwitdh (MB)
Whitewash	95.57	23.56	941.15	241.02
CMTB	453.36	41.05	1,335.75	374.03
This work	**3.29**	**0.49**	**24.97**	**4.24**

only one that allows authentication every 60 seconds or less). The results in tables IV and V also indicate that our approach leads to a substantial reduction in terms of bandwidth usage compared to the state of the art. Our protocol reduces the overall amount of data exchanged between the parties by one to two orders of magnitude, and requires between 0.06MB and 4.24MB depending on the circuit. Moreover, the amount of data exchanged by the smartphone is very small, accounting for just a fraction of the overall communication.

In terms of energy, the cost of traditional garbled circuits secure in the malicious model is clearly unsustainable on smartphones (see Table IV). With scaled Manhattan distance computation, the smartphone is able to run between 35 and 123 authentication attempts before exhausting the battery. This corresponds to half-hour to two hours of smartphone usage with 60-second authentication windows, and assuming that the smartphone is not performing any other task. Similarly, when using Hamming distance, the energy provided by the battery is sufficient for performing up to 7 authentication attempts with 1,600 bits of input, and none using 16,384 bits.

In comparison, the energy cost of our approach is very small. Our technique requires less than 0.2 mWh for computing scaled Manhattan distance, and between 1.23 mWh and 12.53 mWh for Hamming distance. Moreover, if the smartphone is allowed to perform some pre-computation (indicated as "offline" in Table IV), the energy cost of scaled Manhattan distance is below 0.05 mWh, and that of Hamming distance is between 0.64 mWh to 6.86 mWh. This corresponds to a negligible impact on the battery life of the device, which allows the user to perform a large number of authentication attempts (from 788 with Hamming distance, to over 1.7M with Manhattan distance) on a single charge.[3]

X. CONCLUSION

In this paper, we presented the first practical energy-efficient outsourced privacy-preserving authentication protocol. Our approach is unique, because it provides security against malicious and possibly colluding adversaries without using cut-and-choose. With our protocol, user authentication is performed in less than one second with scaled Manhattan distance, and in 3.29-24.97 seconds with Hamming distance, which is significantly faster than previous protocols. As a consequence, our technique is currently the only one suitable for continuous smartphone user authentication with windows of 60 seconds or shorter.

Because our protocol targets continuous authentication of smartphone users, we measured energy consumption and reported the overhead of our technique. Our experiments show that the impact on the smartphone's battery life is very small (between 0.05 mWh and 12.53 mWh), and negligible if circuit construction is performed offline, e.g., while the smartphone is charging. Our proposal makes privacy-preserving continuous authentication on smartphones eminently feasible from a computational and energy consumption standpoint.

[3]It is possible that our protocol uses more energy than Whitewash if both online and offline stages are performed on battery power. However: (1) the energy consumption of our protocol is negligible; (2) the energy cost can be further reduced by constructing the circuit offline as reported in Table IV; and (3) the protocol execution time for Whitewash is 95.57-941.15 seconds (compared to 3.29-24.97 seconds), which makes it unsuitable for low-latency authentication. For the aforementioned reasons, we believe that the energy comparison between our protocol and Whitewash is not necessary.

REFERENCES

[1] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *EUROCRYPT 2015*, pages 673–701, 2015.

[2] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith. Smudge attacks on smartphone touch screens. In *4th USENIX Workshop on Offensive Technologies, WOOT '10*, 2010.

[3] L. Ballard, S. Kamara, F. Monrose, and M. Reiter. Towards practical biometric key generation with randomized biometric templates. In *CCS*, 2008.

[4] L. Ballard, S. Kamara, and M. Reiter. The practical subtleties of biometric key generation. In *USENIX Security Symposium*, 2008.

[5] D. Beaver. Server-assisted cryptography. In *Proceedings of the 1998 Workshop on New Security Paradigms*, pages 92–106, 1998.

[6] M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS*, pages 190–209, 2011.

[7] J. Boyar and R. Peralta. The exact multiplicative complexity of the hamming weight function. 2003.

[8] H. Carter, C. Lever, and P. Traynor. Whitewash: outsourcing garbled circuit generation for mobile devices. In *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC*, 2014.

[9] H. Carter, B. Mood, P. Traynor, and K. Butler. Secure outsourced garbled circuit evaluation for mobile devices. In *USENIX Security*, 2013.

[10] R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Proc. of the Intl. Conf. on the Theory and Application of Cryptographic Techniques*, 2001.

[11] Data genetics: Pin analysis. http://www.datagenetics.com/blog/september32012/. Accessed: 2015-01-08.

[12] M. Derawi, C. Nickel, P. Bours, and C. Busch. Unobtrusive user-authentication on mobile phones using biometric gait recognition. In *Intelligent Information Hiding and Multimedia Signal Processing*, 2010.

[13] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D.Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *TIFS*, 8(1), 2013.

[14] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[15] C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *EUROCRYPT*, 2011.

[16] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[17] V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In *EUROCRYPT*, 2008.

[18] Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *CRYPTO*, 2013.

[19] A. K. Jain, R. Bolle, and S. Pankanti. *Biometrics: personal identification in networked society*. Springer, 1999.

[20] A. Juels and M. Wattenberg. A fuzzy commitment scheme. In *CCS*, 1999.

[21] S. Kamara, P. Mohassel, M. Raykova, and S. S. Sadeghian. Scaling private set intersection to billion-element sets. In *Financial Cryptography and Data Security*, pages 195–215, 2014.

[22] S. Kamara, P. Mohassel, and B. Riva. Salus: a system for server-aided secure function evaluation. In *ACM CCS'12,*, pages 797–808, 2012.

[23] K. Killourhy and R. Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. In *Proc. of the Annual IEEE/IFIP Intl. Conf. on Dependable Systems and Networks*, 2009.

[24] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS*, 2009.

[25] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming*, 2008.

[26] B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security*, pages 285–300, 2012.

[27] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO 2013*, pages 1–17. Springer, 2013.

[28] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of cryptology*, 25(4), 2012.

[29] Y. Lindell and B. Riva. Cut-and-choose yao-based secure computation in the online/offline and batch settings. In *Advances in Cryptology–CRYPTO 2014*, pages 476–494. Springer, 2014.

[30] H. Lu, A. J. B. Brush, B. Priyantha, A. K. Karlson, and J. Liu. Speakersense: Energy efficient unobtrusive speaker identification on mobile phones. In *Pervasive Computing 2011,*, pages 188–205, 2011.

[31] F. Monrose, M. Reiter, Q. Li, and S. Wetzel. Cryptographic key generation from voice. In *IEEE S&P*, 2001.

[32] F. Monrose, M. Reiter, and S. Wetzel. Password hardening based on keystroke dynamics. *Int. J. Inf. Sec.*, 1(2):69–83, 2002.

[33] Monsoon power monitor. http://www.msoon.com/LabEquipment/PowerMonitor/. Accessed: 2015-08-14.

[34] A. Nagar, K. Nandakumar, and A. K. Jain. Biometric template transformation: a security analysis. In *Media Forensics and Security II, IS&T-SPIE Electronic Imaging Symposium*, page 75410, 2010.

[35] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium On Discrete Algorithms (SODA)*, 2001.

[36] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. SCiFI – A system for secure face identification. In *IEEE S&P*, 2010.

[37] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *ASIACRYPT*, 2009.

[38] T. Plantard, W. Susilo, and Z. Zhang. Fully homomorphic encryption using hidden ideal lattice. *TIFS*, 8(12), 2013.

[39] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC*, 1989.

[40] K. A. Rahman, K. S. Balagani, and V. V. Phoha. Snoop-forge-replay attacks on continuous verification with keystrokes. *TIFS*, 8(3), 2013.

[41] N. K. Ratha, J. H. Connell, and R. M. Bolle. Enhancing security and privacy in biometrics-based authentication systems. *IBM Systems Journal*, 40(3):614–634, 2001.

[42] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *Intl. Conf. on Information Security and Cryptology*, 2009.

[43] A. Serwadda and V. Phoha. When kids toys breach mobile phone security. In *ACM CCS*, 2013.

[44] A. Serwadda, V. Phoha, and Z. Wang. Which verifiers work?: A benchmark evaluation of touch based authentication algorithms. In *BTAS*, 2013.

[45] A. Serwadda and V. V. Phoha. Examining a large keystroke biometrics dataset for statistical-attack openings. *ACM Trans. Inf. Syst. Secur.*, 16(2):8, 2013.

[46] A. Shelat and C. Shen. Fast two-party secure computation with minimal assumptions. In *2013 ACM CCS'13*, pages 523–534, 2013.

[47] A. shelat and C.-h. Shen. Two-output secure computation with malicious adversaries. In *EUROCRYPT*, volume 6632, pages 386–405. 2011.

[48] E. Shi, Y. Niu, M. Jakobsson, and R. Chow. Implicit authentication through learning user behavior. In *Info. Security Conference*, 2010.

[49] D. Shukla, R. Kumar, A. Serwadda, and V. V. Phoha. Beware, your hands reveal your secrets! In *ACM CCS*, pages 904–917, 2014.

[50] Z. Sitová, J. Šeděnka, Q. Yang, G. Peng, G. Zhou, P. Gasti, and K. Balagani. HMOG: A New Biometric Modality for Continuous Authentication of Smartphone Users. In *TIFS*, 2015.

[51] J. Šeděnka, S. Govindarajan, P. Gasti, and K. Balagani. Secure outsourced biometric authentication with performance evaluation on smartphones. *TIFS*, 8(1), 2014.

[52] Y. Xu, J. Heinly, A. M. White, F. Monrose, and J. Frahm. Seeing double: reconstructing obscured typed input from repeated compromising reflections. In *2013 ACM CCS'13*, pages 1063–1074, 2013.

[53] A. Yao. How to generate and exchange secrets. In *FOCS*, 1986.

[54] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*. 2015.