Privacy Preserving Biometric Authentication for Fingerprints and Beyond

Marina Blanton
University at Buffalo
Buffalo, USA
mblanton@buffalo.edu

Dennis Murphy
University at Buffalo
Buffalo, USA
dpm29@buffalo.edu

ABSTRACT

Biometric authentication eliminates the need for users to remember secrets and serves as a convenient mechanism for user authentication. Traditional implementations of biometric-based authentication store sensitive user biometry on the server and the server becomes an attractive target of attack and a source of large-scale unintended disclosure of biometric data. To mitigate the problem, we can resort to privacy-preserving computation and store only protected biometrics on the server. While a variety of secure computation techniques is available, our analysis of privacy-preserving biometric authentication constructions revealed that available solutions fall short of addressing the challenges of privacy-preserving biometric authentication. Thus, in this work we put forward new constructions to address the challenges.

Our solutions employ a helper server and use strong threat models, where a client is always assumed to be malicious, while the helper server can be semi-honest or malicious. We also determined that standard secure multi-party computation definitions are insufficient to properly demonstrate security in the two-phase (enrollment and authentication) entity authentication application. We thus extend the model and formally show security in the multi-phase setting, where information can flow from one phase to another and the set of participants can change between the phases. We implement our constructions and show that they exhibit practical performance for authentication in real time.

KEYWORDS

secure computation, biometric authentication, multi-phase secure execution, garbled circuit evaluation, oblivious transfer

1 INTRODUCTION

Biometric-based authentication provides a convenient user authentication mechanism which does not require users to remember passwords or maintain other secrets. Biometric-based authentication is also now more easily accessible than before to the average user for a variety of application due to proliferation of smartphones equipped with sufficient sensors. Biometric data, however, requires strong protection because, unlike password-based authentication, biometry cannot be replaced if the data becomes compromised. Enhancing protection of biometric data used in biometric-based authentication is the focus of this work.

We consider the problem of privacy-preserving biometric authentication in a system where users authenticate to a server using their biometric data, but the authentication server does not have access to the users' biometric data in the clear. If the information stored on the server does not allow one to recover user's biometric samples, user biometric data cannot be easily abused by insiders or through computer break-ins. Large-scale leakage of sensitive biometric data is of growing concern due to increasing availability of large-scale biometric data sets. Thus, this work targets designing a robust and practical solution to privacy-preserving biometric-based authentication which can be employed in place of traditional biometric-based authentication mechanisms and which makes abuse of sensitive biometric data more difficult.

In the context of privacy-preserving biometric-based authentication, we can consider two types of solutions: (i) those based on secure sketches and fuzzy extractors and (ii) solutions based on secure multi-party computation (SMPC). The former has a disadvantage that it discloses partial information about each biometric sample, the implications of which are hard to quantify, and we focus on the latter that can guarantee that no biometric-related information of a user is disclosed to any party.

Now if we consider secure two-party computation between a user and an authentication server, we can distinguish between two types based on the amount of interaction: (i) interactive two-party computation where the user carries the full burden of participating in secure evaluation of biometric matching and (ii) non-interactive computation on encrypted data. Note that in the context of user authentication, we must assume that a user can act maliciously in the attempt to circumvent the authentication mechanism and obtain access to the system at any cost. This means that when modeling SMPC, we must provide security in the presence of malicious users, which increases the protocols' cost. This is undesirable for clients operating from computationally-limited battery-powered devices and thus may present usability concerns. On the other hand, non-interactive computation that employs fully homomorphic encryption (HE) permits comparing an encrypted biometric sample provided by the user at authentication time and the encrypted sample captured at the enrollment time. A concern with this solution is that, in order for the user biometric data to stay private from the server, the decryption key must be available only to the client. This means that it is not possible for a client to enroll once and later be able to authenticate from any computer or device because each device has to have the user's private decryption key. This nullifies the advantages of biometric-based authentication which permits authentication without the need to remember passwords or maintain secret keys and brings us back to requiring the user to use secrets together with their biometry.

To mitigate these issues, our approach is to introduce a helper server. This is not a new idea by itself, but it makes a significant difference for this application. The helper server does not contribute any inputs and does not learn any information about user biometric data or the result of user authentication, but rather contributes its

computational power and can store protected biometric data. Multiple authentication servers can use the same helper server. This setup improves both usability and efficiency, as we demonstrate in this work. In particular, that expands the set of techniques we can use for privacy-preserving biometric matching and authentication and consequently aids efficiency. It also permits minimal involvement of users and removes the need for storing any keys or other secrets on user devices. This improves usability and enables a user to authenticate from different devices and a variety of platforms including weak battery-powered devices.

An interesting aspect of this work is that we found that employing traditional SMPC security definitions for (privacy-preserving) authentication is insufficient and there is a need for new definitions. In particular, SMPC is concerned with a single evaluation of a function, during which the set of participants does not change. In the context of authentication, on the other hand, we deal with two phases: enrollment and authentication. Furthermore, the participants themselves can change because a malicious user might attempt to impersonate another user during authentication (while enrollment was carried out by the authentic user). While we can use traditional security definitions to ensure that the participants do not learn unauthorized information during a given phase, there is still a need to link the two phases together and ensure that no biometric-based information is available to the participants as a result of information flow from one phase to another. This is because the servers will obtain certain output after the enrollment phrase, but the output of function evaluation is never protected under the standard definition and is not treated as leakage. Thus, in this application we need to consider the overall view of the two-step process, conceptually treating the output of the first stage as an intermediate result (which must reveal no information) and not as the target output (which is allowed to reveal information). This will also permit us to demonstrate that a malicious user is unable to learn sensitive biometric data of any enrolled user.

We determined that a few prior publications that treat the topic of biometric-based authentication [1, 3, 20] use a two-phase model; however, the definitions have custom interfaces and are not applicable to other functionalities. We provide a more detailed comparison in the related work section.

Our solution is based on garbled circuit evaluation (GCE) [36] and we use two strong threat models, in both of which the client is malicious and can behave arbitrary. In the weaker model, the servers are semi-honest (follow the prescribed protocols) and do not collude with each other or the clients. In the stronger model, the helper server can act maliciously and can additionally collude with clients. When building our constructions, we introduce a variant of oblivious transfer (OT), termed oblivious transfer with bit operations (OTB), which may be of independent interest, and consider over-the-threshold cosine similarity and Euclidean distance as the basis for biometric matching. We formally prove security of our solutions under standard security definitions, expanded as discussed above to accommodate multi-phase computation where the participants can change between the phases. We also implement and empirically evaluate performance of our solutions and show that they are well suited for authentication in real time.

Related Work. The first line of work that employ cryptography to protect confidentiality of biometric data during matching uses secure sketches and fuzzy extractors, e.g., [10, 23, 28, 30, 35], some of which make use of an additional secret or password to improve the properties of the solution. The second line of research – closer to this work – uses SMPC or secure outsourcing. Constructions for different biometric modalities have been developed. For instance, they include face [14, 19, 31, 33], iris [11, 12], fingerprints [7, 12, 13, 18], voice [4], and others. The computation itself widely differs in the complexity, ranging from simple Hamming or Euclidean distance over integers to hidden Markov model evaluation on floating-point values. A variety of techniques have been used including GCE, secret sharing, encryption with special properties (e.g., HE and predicate encryption), and a combination thereof. Most results above above focused on privacy-preserving biometric matching or identification in the semi-honest security setting. Authentication, however, demands a stronger security model in which clients must be assumed to be malicious.

Publications that treat privacy-preserving biometric authentication in the presence of a malicious client include [1, 2, 16, 21, 34][1]. Several of them use HE to perform a simple distance computation and disclose it to one of the parties. For instance, in [15, 16, 21] the server computes $\text{dist} \cdot r_0 + r_1$, where dist is (Hamming or Euclidean) distance between the enrollment and current biometric samples and r_0, r_1 are large random values. The use of the randomizing values prevents a malicious client from making meaningful changes to the distance prior to sending it to the server. This structure has two disadvantages: (i) each client has to maintain a secret key on each device he/she wants to use for authentication (which our solution is set to mitigate) and (ii) the computed distance is revealed to one of the parties, commonly the server who can compile distributions of this information for each user over time or, worse, to the (malicious) client who can use the distance as the guide for improving its strategy for impersonating the authentic user. [34] employs HE in their semi-honest protocol and GCE when the client can be malicious. In both cases, the client has to maintain a secret key or another state. Further, none of these solutions connect enrollment and authentication to protect enrollment data from malicious users.

HE together with digital signatures, GCE, and zero-knowledge proofs are used in [2] for evaluating cosine similarity as a distance metric. While it does treat malicious adversaries, active tampering is limited to client devices only, and the remaining participants are semi-honest. Specifically, these participants are service providers and terminals, where a terminal is an external device outside client control which obtains the authentication phase biometric. This setting is equivalent to our weaker (non-colluding) security model. Both secret keys and encoded templates are stored on the devices, which can reveal non-revocable template information to adversaries with access to the device. Two of their constructions also leak the computed distance. [1] uses the same set of primitives in a more general construction proven UC-secure, where multiple devices need to interact to authenticate the client. However, secret keys are distributed across devices such that an adversary controlling

[1] In addition, [29, 37] are also said to provide privacy-preserving authentication. However, in the solution of [37] an authority obtains cleartext access to user biometrics and thus does not achieve privacy, while in [29] the client is considered fully trusted and consequently the construction does not correspond to authentication.

enough of them gains the secret key along with non-revocable enrollment templates. The protocol is for cosine similarity and proved secure in the random oracle model. [25] is a general-purpose SMPC compiler which can support a similar structure but cannot handle collusion between active adversaries and does not consider participants which may change between phases. A recent work [20] proposed a dynamic and multi-phase protocol based on functional encryption. It, however, still requires the client to store secrets and the performance is slower than ours, despite not taking network communication time into account as we do.

As far as definitional differences go, [3] provides custom interfaces, algorithms, and security properties and does not use standard SMPC definitions. [2] uses the real-ideal paradigm, but does not discuss the possibility of the enrollment and matching phases being carried out by different parties. [1] considers the UC framework, but does not provide general SMPC definitions.

There are publications that modify conventional matching for a biometric modality to be more amenable for use with privacy-preserving techniques. A notable example is SCiFI [31] that designed a new face identification mechanism and built a corresponding privacy-preserving protocol. Fingerprints are another example where conventional minutiae-based matching is complex, and thus privacy-preserving solutions initially focused on simple but inaccurate FingerCode matching [7], and eventually grew to support conventional minutiae-based matching [8]. A more attractive approach is to develop a new feature representation and matching algorithm of high accuracy, as was done in DeepPrint [17]. That work showed that it is possible to represent fingerprints as fixed-length vectors and use simple Euclidean distance or cosine similarity for biometric matching, while achieving nearly identical accuracy to that of top performing variable-length minutiae-based matching algorithms. We built on [17] and use it as a basis for biometric matching in secure authentication protocols we develop. For potential deployment at scale, [27] provides an efficient virtual memory manager for secure computation and may facilitate efficient batching of many authentication sessions in parallel.

2 PRELIMINARIES

2.1 Problem Statement

We consider a setting where a client C uses a service that employs biometric data for entity authentication. At the enrollment time, the client registers with the service, which involves C capturing its biometric sample, extracting features to produce representation B, and storing the result in a privacy-preserving way with the service. At the time of authentication, the client captures a new biometric sample and produces representation $\widehat{B}$, after which the client and the service engage in a protocol. As a result, the client is either authenticated and gains access to the service or is denied access. The computation involves performing biometric matching by first computing the distance between the enrollment and current biometric samples, $d = \mathrm{dist}(B, \widehat{B})$, and consequently comparing the distance to a predefined threshold t.

Because we utilize a helper server for usability and efficiency reasons, we denote the main authentication server as S_1 and a helper server as S_2 (recall that the same helper server can be employed by different services). Security requirements are such that S_1 has no access to sensitive biometric information about any user C and only determines the outcome of each authentication (i.e., whether the supplied biometric was a close match and is considered authentic using the over-the-threshold computation described above). S_2 learns no information about any biometrics and no information about authentication outcomes, i.e., its purpose is to improve efficiency and usability of the protocols for the client and the service.

Because we work with authentication, we must assume that the client is malicious, i.e., it will try all means at its disposal in the attempt to successfully authenticate without sufficient credentials. The servers, on the other hand, can be more trustworthy and can be expected to follow the prescribed computation. In particular, because S_1 is the authentication server, it must properly enforce access control and correctly perform the computation (as otherwise no meaningful guarantees can be maintained in the presence of a malicious client). However, someone with access to the server (e.g., a dishonest insider or in the case of a computer break-in) might be interested in extracting biometric information about the users from the information that the server handles. This includes information stored at the server and the server's view during all registration and authentication protocols. For that reason, we begin with a model of the servers being semi-honest and non-colluding.

In addition, because the helper server S_2 is not controlled by the service and may not be as trustworthy, we consider the possibility of S_2 behaving maliciously. For that reason, we consider a stronger security model, in which S_1 remains to be semi-honest and non-colluding with other parties, while S_2 can be malicious and possibly colluding with clients C (who are always assumed to be malicious). This stronger model has implications on the cost of the protocols in order to maintain security.

2.2 Security Definitions

We use the standard formulation of security that relies on the real/ideal paradigm in the presence of malicious adversaries and guarantees correctness and no unintended information disclosure. The definition requires that the view of any adversary in real protocol execution is computationally indistinguishable from its view in an ideal world execution, where an ideal functionality produces the output and parties not controlled by the adversary are not participating in the computation.

In a general setup, let parties $P_1, \ldots, P_n$ engage in a secure multiparty protocol Π that computes function f. We specify f as taking n inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and producing n outputs $\mathbf{y}_1, \ldots, \mathbf{y}_n$, i.e., $f(\mathbf{x}_1, \ldots, \mathbf{x}_n) = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$. Each $\mathbf{x}_i$ and $\mathbf{y}_i$ is treated as a vector to permit entering and receiving multiple values, but some participants may not provide any inputs and/or receive no output (in which case the corresponding $\mathbf{x}_i$ and/or $\mathbf{y}_i$ is empty).

Adversary $\mathcal{A}$ is permitted to corrupt one or more participants based on the threat model. The remaining parties are honest and denoted by $\mathcal{H}$. We let $VIEW_{\Pi,\mathcal{A}}$ denote the view of adversary $\mathcal{A}$ after an execution of Π. The view is the union of the views of the parties controlled by $\mathcal{A}$, which include their inputs, randomness used during the computation, and all messages received during the computation from other participants. We also let $OUT_{\Pi,\mathcal{H}}$ denote the output of the honest parties after the execution, i.e., the produced $\mathbf{y}_i$ that correspond to the honest parties. Let κ denote a

security parameter and define

$$REAL_{\Pi,\mathcal{A}}(1^\kappa, \{\mathbf{x}_i\}_{i=1}^n) \stackrel{\text{def}}{=} VIEW_{\Pi,\mathcal{A}} \cup OUT_{\Pi,\mathcal{H}}$$

In the ideal world, there is no protocol execution and instead a probabilistic polynomial time (PPT) simulator $\mathcal{S}$ interacts with $\mathcal{A}$. The simulator is able to query an ideal functionality $\mathcal{F}$ which computes function f on behalf of the participants and the goal is to simulate Π's execution without access to the data of non-corrupt participants. As before, the view of $\mathcal{A}$ corresponds to the inputs, random choices, and the messages received by the parties controlled by $\mathcal{A}$ during the simulation, which we denote by $VIEW_{f,\mathcal{A}}$.

The ideal functionality evaluates function f on behalf of the participants. It uses inputs of honest participants and obtains inputs of corrupt participants from $\mathcal{S}$. When $\mathcal{A}$ is semi-honest, $\mathcal{S}$ obtains access to inputs of the corrupt parties controlled by $\mathcal{A}$ and supplies them to $\mathcal{F}$. When $\mathcal{A}$ is malicious, it can instruct the parties it controls to deviate from the prescribed computation and enter their inputs into the computation in a different form. Thus, it is $\mathcal{S}$'s task to extract the corrupt parties' inputs the way they are entered into the computation and communicate the inputs to $\mathcal{F}$, who will evaluate the function using the supplied inputs. Note that $\mathcal{S}$ or $\mathcal{F}$ can abort if either of them obtain empty or malformed inputs or messages. If the evaluation is successful, the parties obtain the output of f, and we denote the output of honest parties by $OUT_{f,\mathcal{H}}$. Similar to the real execution, we define

$$IDEAL_{f,\mathcal{S}(\mathcal{A})}(1^\kappa, \{\mathbf{x}_i\}_{i=1}^n) \stackrel{\text{def}}{=} VIEW_{f,\mathcal{A}} \cup OUT_{f,\mathcal{H}}$$

Given the above, we formulate the security definition as:

Definition 1. *An n-party protocol Π between $P_1, \ldots, P_n$ securely evaluates function f if for all PPT adversaries $\mathcal{A}$ controlling a subset of the participants, all input vectors $\mathbf{x}_i$, and $\kappa \in \mathbb{Z}$, there exists a PPT simulator $\mathcal{S}$ such that*

$$REAL_{\Pi,\mathcal{A}}(1^\kappa, \{\mathbf{x}_i\}_{i=1}^n) \stackrel{c}{\approx} IDEAL_{f,\mathcal{S}(\mathcal{A})}(1^\kappa, \{\mathbf{x}_i\}_{i=1}^n)$$

where $\stackrel{c}{\approx}$ denotes computational indistinguishability.

Because in our context information produced in one phase of the computation is used as input into another phase, we extend the standard definition to support multi-stage computation. For simplicity, we consider computation consisting of two phases, but the concept easily generalizes to any number of phases. To accomplish this, we define the outputs of the first phase to be additional, auxiliary inputs $\mathbf{u}_i$ (which may be empty) into the second phase. Conceptually, this can be pictured as we pause after the first phase, save the output as the current state, and resume the computation once the inputs into the second phase are received. Note that each phase receives inputs from the parties and the second phase additionally receives the outputs from the first phase in the form of auxiliary inputs.

It is important to take into account that the participating parties might change between the phases of the computation. This is the case for authentication applications, where a malicious user (imposter) might attempt to authenticate impersonating another user who previously enrolled in the system (authentic user). For that reason, we define two different, overlapping sets of participants $P_1^{(1)}, P_2^{(1)}, \ldots, P_{n_1}^{(1)}$ and $P_1^{(2)}, P_2^{(2)}, \ldots, P_{n_2}^{(2)}$. Here superscript (j) denote data associated with phase j and n_1 (respectively, n_2) denote the number of participants in the first (resp., second) phase. If $P_i^{(1)} = P_j^{(2)}$ for some i and j, i.e., the party is involved in both phases, then it will have an auxiliary input for the second phase. The remaining participants, i.e., those who are involved only in one of the phases, contribute their input and receive the output as in the conventional formulation of an execution.

The more complex participant structure requires that we also carefully specify adversarial corruptions. If a party is controlled by an adversary, the adversary controls it in both stages of the computation. If an adversary controls multiple conspiring participants, it will control them in all phases in which the parties are active protocol participants.

Let f denote the multi-phase functionality and $f^{(1)}$ and $f^{(2)}$ denoted the functions we evaluate in phases 1 and 2, respectively. The auxiliary input is set for each $P_i^{(2)}$ as $\mathbf{u}_i^{(2)} = \mathbf{y}_j^{(1)}$ if $P_i^{(2)} = P_j^{(1)}$ for some j and $\mathbf{u}_i^{(2)}$ is empty if $P_i^{(2)}$ was not a protocol participant in phase 1. Given this, we define real and ideal views in the second (or any subsequent) phase of the computation as $REAL_{\Pi^{(2)},\mathcal{A}}(1^\kappa, \{\mathbf{x}_i^{(2)}, \mathbf{u}_i^{(2)}\}_{i=1}^{n_2}) \stackrel{\text{def}}{=} VIEW_{\Pi^{(2)},\mathcal{A}} \cup OUT_{\Pi^{(2)},\mathcal{H}}$ and $IDEAL_{f^{(2)},\mathcal{S}(\mathcal{A})}(1^\kappa, \{\mathbf{x}_i^{(2)}, \mathbf{u}_i^{(2)}\}_{i=1}^{n_2}) \stackrel{\text{def}}{=} VIEW_{f^{(2)},\mathcal{A}} \cup OUT_{f^{(2)},\mathcal{H}}$.

Definition 2. *A sequence of two protocols $\Pi^{(1)}$ and $\Pi^{(2)}$, executed by parties $P_1^{(1)}, \ldots, P_{n_1}^{(1)}$ and $P_1^{(2)}, \ldots, P_{n_2}^{(2)}$, respectively, securely evaluates the sequence of functions $f^{(1)}$ and $f^{(2)}$ if for all PPT adversaries $\mathcal{A}$ controlling a subset of the parties, all input vectors $\mathbf{x}_i^{(1)}$, $1 \le i \le n_1$, and $\mathbf{x}_i^{(2)}$, $1 \le i \le n_2$, all auxiliary input vectors $\mathbf{u}_i^{(2)} = \mathbf{y}_j^{(1)}$ subject to $P_i^{(2)} = P_j^{(1)}$, and $\kappa \in \mathbb{Z}^+$, there exists PPT simulator $\mathcal{S}$ such that*

$$REAL_{\Pi^{(1)},\mathcal{A}}(1^\kappa, \{\mathbf{x}_i^{(1)}\}_{i=1}^{n_1}) \stackrel{c}{\approx} IDEAL_{f^{(1)},\mathcal{S}(\mathcal{A})}(1^\kappa, \{\mathbf{x}_i^{(1)}\}_{i=1}^{n_1}) \text{ and}$$

$$REAL_{\Pi^{(2)},\mathcal{A}}(1^\kappa, \{\mathbf{x}_i^{(2)}, \mathbf{u}_i^{(2)}\}_{i=1}^{n_2}) \stackrel{c}{\approx} IDEAL_{f^{(2)},\mathcal{S}(\mathcal{A})}(1^\kappa, \{\mathbf{x}_i^{(2)}, \mathbf{u}_i^{(2)}\}_{i=1}^{n_2}).$$

For the purposes of this work, the computation participants are C, S_1, and S_2, i.e., we are dealing with three-party computation. As described earlier, we consider two threat models:

(1) The minimal meaningful security model that treats C as malicious and S_1 and S_2 as semi-honest and non-colluding. For the purposes of showing security, this means that $\mathcal{A}$ can corrupt one party at a time with the specified semi-honest/malicious abilities and our solutions need to be secure for each instantiation of $\mathcal{A}$.
(2) A stronger security model in which, in addition to malicious client C, helper server S_2 can behave maliciously and collude with some clients. Recall that it is not meaningful to assume that S_1 is malicious in the context of this application, and S_1 also does not collude with other parties. This means that $\mathcal{A}$ has two instantiations: semi-honest S_1 and malicious and colluding C and S_2.

The user who participates in the registration is called authentic C_{auth}. The same or a different user might attempt to authenticate later by engaging in the authentication protocol. If the user does

not change, the parties $S_1, S_2, C_{\text{auth}}$ participate in both phases of the protocol. Otherwise, the second phase is a three-party protocol executed by S_1, S_2, and an imposter client, denoted as C_{imp}.

2.3 Building Blocks

In this work, we use the following cryptographic primitives:

- *Oblivious Transfer (OT)* is a protocol between two parties, sender S and receiver R. In 1-out-of-2 OT, OT_1^2, the sender holds two strings, m_0 and m_1, while the receiver holds bit b and learns m_b. The security requirements are that the sender learn nothing about b, while the receiver learns nothing about the remaining string m_{1-b}.
  Additionally, we employ a new (to the best of our knowledge) generalization of OT, which we refer to as *Oblivious Transfer with Bit Operations (OTB)*. In this setting, the sender additionally holds an input bit c, and the receiver obtains $m_{b \odot c}$, where $\odot$ is a previously agreed upon boolean binary function. Details are provided in Section 2.3.1.
  OT extensions are commonly used for efficiency when multiple calls to OT are needed. OT and OT extensions take a computational security parameter κ, and constructions secure in the malicious model can also rely on a statistical security parameter ρ.
- *Garbled circuit (GC) evaluation* is a secure two-party protocol parameterized by computational security parameter κ that evaluates some function f, represented as a boolean circuit, on private inputs. One party, garbler G, chooses two random labels ℓ_i^0, ℓ_i^1 to represent each (boolean) wire i in the circuit. For each binary gate of the circuit, G derives an encryption key from each of the four possible input wire label pairs and uses these to encrypt the label of the corresponding output wire. This collection of per-gate tables constitutes the garbled circuit $\mathcal{G}_f$. The other party, evaluator E, receives from G both $\mathcal{G}_f$ and the set of input wire labels corresponding to G's input values (which are required to not reveal anything about the input they represent). E then engages in an OT_1^2 protocol with G to obtain the wire labels corresponding to E's input values. Finally, E evaluates the circuit gates beginning with the input labels and obtains the final output label(s). At the end of the protocol, E sends the corresponding output label(s) to G (which necessarily reveals the actual output to G). For the construction to comply with the security definition, it must be the case that
  - G and E learn nothing about each others' input and
  - G learns the function output.

  The literature contains a number of well known optimizations to the original Yao construction [36]. This includes the use of the "free XOR" gates introduced in [26] which imposes a certain relationship between the two labels corresponding to a wire, namely, that $\ell_i^1 \oplus \ell_i^0 = \Delta$ for each wire i. The labels are also commonly generated as pseudo-random strings. In our implementation discussed in Section 4, we use garbling as in the JustGarble work [9].
  The conventional variant of GCE for semi-honest adversaries provides resilience against malicious evaluators, as long as the appropriate variant of OT is used. We do not require a strengthened variant secure against malicious adversaries, since within our protocols it is possible to arrange for the circuit garbler to be semi-honest.
- A *commitment scheme* is parameterized by a security parameter κ and characterized by two algorithms, commit and open. The commit algorithm is randomized and denoted by $c = \text{com}(x, r)$, where x is the value being committed and r is randomness specified explicitly. We call c to be a commitment to x. Commitment c can later be opened (typically by revealing x and r), which exposes the value of x. The security requirements are hiding and biding properties of the commitment scheme. Namely, hiding requires that the release of c does not disclose information about x and binding requires that it is infeasible to open a commitment c to any value other than the value x used to produce the commitment. The security guarantees can be information-theoretic or computational.

2.3.1 Oblivious Transfer with Bit Operations. This generalization of OT works with any already proven secure OT scheme. Here, the parties agree upon a binary boolean function, denoted as $\odot : \mathbb{F}_2 \times \mathbb{F}_2 \to \mathbb{F}_2$. In addition to the sender holding messages m_0 and m_1 and the receiver holding bit b, the sender now also holds an input bit c. Then the receiver obtains $m_{b \odot c}$ without learning anything else, while the sender learns nothing about receiver's input b.

This operation is realized using regular OT, where instead of entering (m_0, m_1), the sender enters $(\widehat{m}_0, \widehat{m}_1)$ specified as follows for the three most common binary boolean operations:

- AND: the sender sets $\widehat{m}_0 = m_0$ and $\widehat{m}_1 = m_c$
- OR: the sender sets $\widehat{m}_0 = m_c$ and $\widehat{m}_1 = m_1$
- XOR: the sender sets $\widehat{m}_0 = m_c$ and $\widehat{m}_1 = m_{\neg c}$

In terms of correctness, notice that in the case of AND, if the sender holds $c = 0$, then the receiver obtains m_0 regardless of their input ($b \wedge 0 = 0$), and m_b otherwise ($b \wedge 1 = b$). Similarly, in the case of OR, if the sender holds $c = 1$, then the receiver obtains m_1 regardless of their input and m_b otherwise. And in the case of XOR, $b \oplus c = c$ if and only if $b = 0$, while $b \oplus c = \neg c = c \oplus 1$ if and only if $b = 1$.

In terms of security, nothing in this modification allows the sender to learn b if the OT being used already prevents this (although the sender may know which string the receiver gets; this is a function of the sender's input). Similarly, the receiver receives exactly one string $m_{b \odot c}$ without learning $m_{\neg(b \odot c)}$ and does not learn anything about c (but the receiver may know they are getting m_b based on their input and the function $\odot$ being computed).

We will be using this OT variant while transferring GC labels that correspond to an XOR-share of the clients' private biometric data and denote it as OTB.

2.4 DeepPrint Fingerprint Matching

We are interested in supporting authentication based on popular biometric modalities with good distinguishing properties such as fingerprints and iris codes. Iris codes are represented as binary strings and their matching is based on the Hamming distance. As a result, iris matching does not introduce significant complexity. On the other hand, conventional minutiae-based comparison of fingerprints is complex and not well suited for use in secure computation.

For that reason, DeepPrint [17] that uses deep neural network for fingerprint feature selection with excellent discriminating properties is of interest to us. The resulting fingerprint representations are fixed length and can be compared using simple conventional distance metrics, making it easier to use the representation with cryptographic tools.

DeepPrint encodes a fingerprint biometric as a vector of 192 single-precision floating-point values, which is normalized to be unit length. A unit-length vector is defined as having its L^2 norm, denoted by $\|B\|$ for a biometric vector B, be equal to 1. Concretely, for vector $B = (B[i])_{i=1}^{w}$, it is required that $\|B\| = \sqrt{\sum_{i=1}^{w} B[i]^2} = 1$.[2] Then the distance between two unit-length DeepPrint representations B and $\widehat{B}$ can be determined using the cosine similarity between the two vectors, defined as the dot product of the vectors divided by the product of their L^2 norms $(\sum_{i=1}^{w} B[i]\widehat{B}[i])/(\|B\|\|\widehat{B}\|)$. Of course, when normalizing to unit length, this division is unnecessary.

The range of values the cosine similarity distance metric may take on normalized inputs is $[-1, 1]$, with 1 representing an exact match. Thus, to determine if two representations are within a close distance, treated as a "match," it suffices to determine if their dot product is within the range $(1 - t, 1]$ for a desired threshold value t.

The authors of [17] also used Euclidean distance as a distance function. For two unit-length vectors B and $\widehat{B}$, Euclidean distance defined as $\sqrt{\sum_{i=1}^{w}(B[i] - \widehat{B}[i])^2}$ yields values in the range $[0, 2]$, with 0 representing an exact match. Thus, a match is determined by checking if the distance is within $[0, t)$ for some threshold t. For performance reasons, we work with squared Euclidean distance, in which case the threshold t needs to be adjusted accordingly. We use notation $d \sim t$ to denote the result of comparing the distance d to threshold t, where the exact operation depends on the distance metric (i.e., checking $1-t < d \leq 1$ for cosine similarity and $d < t$ for Euclidean distance); $\text{dist}(B, \widehat{B})$ denotes the distance computation.

DeepPrint representation requires 768 bytes of storage for 192 32-bit (single precision) floating-point values, but can be compressed to 200 bytes. This is accomplished in [17] by compressing a floating-point vector element to an 8-bit integer using min-max normalization as follows: Given DeepPrint floating-point vector $B = (B[1], \ldots, B[192])$, define $h_B = \max_i\{B[i]\}$, $\ell_B = \min_i\{B[i]\}$ and compute

$$\widehat{B}[i] = \left\lfloor \frac{255(B[i] - \ell_B)}{h_B - \ell_B} \right\rfloor \tag{1}$$

for $i \in [1, 192]$. The compressed representation stores 192 8-bit integers $\widehat{B}[i]$ and two 32-bit floating point values h_B and ℓ_B. Matching of two compressed representations is performed by decompressing the representations and computing the distance on floats. The compression has a minimal impact on the matching accuracy [17].

2.5 Vector Normalization in Adversarial Settings

Normalization of DeepPrint biometric representations is assumed to be performed as part of feature extraction after biometric sampling. Its presence has a direct impact on how the threshold t that determines a match of two biometric representations is chosen: scaling normalization will result in scaling the threshold t as well.

[2]For performance reasons, we can instead check the square of the norm against 1.

This is of interest for us because in the context of this work a biometric sample comes from a user who can act maliciously and construct a biometric representation that deviates from the expectations including normalization. Thus, it becomes important to enforce proper normalization of a biometric representation a user submits. If normalization is not enforced, a malicious user can succeed with authentication without a matching biometric by manipulating vector normalization. As a specific example, consider that squared Euclidean distance is used for distance computation and biometric vectors B are assumed to be unit-length normalized (i.e., $\|B\| = \|B\|^2 = 1$). For two vectors B and $\widehat{B}$, the squared Euclidean distance is $\|B - \widehat{B}\|^2$. By the triangle inequality, we have

$$\|B - \widehat{B}\|^2 \leq (\|B\| + \|\widehat{B}\|)^2 = \|B\|^2 + 2\|B\|\|\widehat{B}\| + \|\widehat{B}\|^2$$

Now if the distance between vectors B and $\widehat{B}$ is compared to a predetermined threshold t and the client is at liberty to normalize both B and $\widehat{B}$ to any value N they wish, then choosing $N < \frac{1}{2}\sqrt{t}$ for both B and $\widehat{B}$ will result in successful authentication independent of the actual vectors (i.e., in such cases, $\|B - \widehat{B}\|^2 < t$ is always true). Even when the adversary tampers with (normalization of) one of the vectors, it is still possible to deviate from the intended authentication rules. For this reason, we enforce proper length normalization of all biometrics and include measures to verify that client submitted biometrics are of the correct form. While there may be input formats and distance metrics which prevent abuse when only the enrollment biometric is properly normalized, we conservatively enforce proper normalization at both enrollment and authentication time.

3 SOLUTIONS BASED ON GARBLED CIRCUIT EVALUATION

Recall that we consider two threat models: (i) semi-honest servers S_1 and S_2 and malicious client and C (ii) semi-honest S_1 and malicious and colluding S_2 and C. We label the first model as SH and the second as MAL. We start with our solution secure in the first model and consequently strengthen it to maintain security in the second, stronger model.

In our solution, the client's involvement is minimal and its task primarily consists of splitting its biometric into two XOR shares and communicating the respective shares to the servers S_1 and S_2. This will take place both at registration and authentication. At registration time, the servers perform the normalization check on the user's private biometric using OTB and GC. In this computation, S_1 acts as the garbler and S_2 as the evaluator. If the normalization check succeeds, the servers accept and store the biometric. The authentication phase proceeds similarly, where in addition to checking whether the submitted biometric meets the normalization criteria, the servers also compute the distance between the registered and newly received biometrics and determine if the distance is within the desired threshold.

When S_2 can be malicious (the second, stronger model), additional information is stored at registration time. In addition to storing shares of user biometric B, the servers obtain and check a one-way function of B that allows the servers to verify correct share reconstruction within the garbled circuit without obtaining

Functionality $\mathcal{F}_{\text{reg-sh}}$

(1) $\mathcal{F}_{\text{reg-sh}}$ receives input $B \in \{0,1\}^m$ from C.
(2) $\mathcal{F}_{\text{reg-sh}}$ samples $r \xleftarrow{R} \{0,1\}^m$ and defines $B_1 = r$ and $B_2 = r \oplus B$.
(3) $\mathcal{F}_{\text{reg-sh}}$ computes $b = (\sum_{i=1}^{m}(B[i])^2 \stackrel{?}{=} 1)$.
(4) $\mathcal{F}_{\text{reg-sh}}$ outputs b to S_1.
(5) If $b = 1$, then $\mathcal{F}_{\text{reg-sh}}$ outputs B_1 to S_1, B_2 to S_2 and accept to C and S_2.
(6) Otherwise, $\mathcal{F}_{\text{reg-sh}}$ outputs $\perp$ to S_1 and S_2 and reject to C and S_2.

Figure 1: Ideal registration functionality with semi-honest servers.

Functionality $\mathcal{F}_{\text{auth-sh}}$

(1) $\mathcal{F}_{\text{auth-sh}}$ receives $B_1 \in \{0,1\}^m$ from S_1, $B_2 \in \{0,1\}^m$ from S_2, and $\widehat{B} \in \{0,1\}^m$ from C.
(2) $\mathcal{F}_{\text{auth-sh}}$ computes $(b_1, b_2) = (\text{dist}(B_1 \oplus B_2, \widehat{B}) \stackrel{?}{\sim} t, \sum_{i=1}^{m}(\widehat{B}[i])^2 \stackrel{?}{=} 1)$.
(3) $\mathcal{F}_{\text{auth-sh}}$ outputs (b_1, b_2) to S_1.
(4) If $(b_1, b_2) = (1, 1)$, then $\mathcal{F}_{\text{auth-sh}}$ outputs accept to C, otherwise $\mathcal{F}_{\text{auth-sh}}$ outputs reject to C.
(5) $\mathcal{F}_{\text{auth-sh}}$ sends terminate to S_2.

Figure 2: Ideal authentication functionality with semi-honest servers.

any information about B. That additional information is used during the authentication phase to ensure that S_2 did not tamper with its values, and we additionally employ stronger tools such as OT resilient to malicious behavior.

In the rest of the paper, we assume a fixed-length biometric representation of m bits (representing w elements of B). Notation $x \xleftarrow{R} X$ means that variable x is sampled uniformly at random from the set X. When working with GCs, we let n denote the total number of wires, where the wires with the lowest indices correspond to the inputs and the wires with the highest indices correspond to the output. The parties hold security parameter κ and agree on the realizations of the building blocks. All protocols assume the existence of secure channels between each pair of parties for sending sensitive information such as shares and keys.

3.1 Malicious C, semi-honest S_1 and S_2

We start the description of our first solution with the expected functionalities for registration and authentication, which are listed in Figures 1 and 2, respectively.

At registration time, the client (which may be corrupt) supplies its biometric B, from which it generates two XOR shares B_1 and B_2. The ideal functionality performs the normalization check for B, the output of which is bit b, which is communicated to S_1. If the check succeeds, the ideal functionality outputs accept to all parties and shares B_1 and B_2 to S_1 and S_2, respectively. Otherwise, the parties receive reject and S_1 and S_2 receive empty string $\perp$ in place of shares.

Protocol 1 Registration Reg-SH

Input: C holds biometric B.
Output: S_1 receives bit b and biometric share B_1; S_2 receives accept or reject and biometric share B_2; C receives accept or reject.
Common Input: Computational security parameter κ.
Protocol steps:

(1) C generates m-bit random value $r \xleftarrow{R} \{0,1\}^m$, sets $B_1 = r$, computes $B_2 = B_1 \oplus B$, and securely communicates B_1 to S_1 and B_2 to S_2. If the receiving server determines that B_1 or B_2 is not an m-bit string, it signals abort.
(2) S_1 generates labels ℓ_i^j for $i \in [1, n]$ and $j \in \{0,1\}$, computes garbled gates $\mathcal{G}_f$ for the normalization check computation, and sends $\mathcal{G}_f$ to S_2.
(3) S_1 and S_2 engage in m instances of OTB_1^2 to communicate to S_2 labels $\ell_i^{B_1[i] \oplus B_2[i]}$ for $i \in [1, m]$: S_1 enters labels $\ell_i^{B_1[i]}$ and $\ell_i^{B_1[i] \oplus 1}$ into OT, S_2 enters bit $B_2[i]$ and learns label $\ell_i^{B_1[i] \oplus B_2[i]} = \ell_i^{B[i]}$.
(4) S_2 evaluates the circuit and sends the computed label of the output wire ℓ_n^b to S_1.
(5) If $\ell_n^b = \ell_n^0$, S_1 signals rejection to C and S_2; S_1 and S_2 output $\perp$.
(6) Otherwise, S_1 signals acceptance to C and S_2; S_1 outputs B_1 and S_2 outputs B_2.

During authentication, servers S_1 and S_2 contribute the shares B_1 and B_2 they received during registration, while the client contributes biometric $\widehat{B}$. The functionality performs two checks:

(1) normalization check for $\widehat{B}$: $b_1 = (\sum_{i=1}^{w}(\widehat{B}[i])^2 \stackrel{?}{=} 1)$
(2) comparison of the distance between enrollment and authentication biometrics B and $\widehat{B}$ to threshold t: $b_2 = (\text{dist}(B_1 \oplus B_2, \widehat{B}) \stackrel{?}{\sim} t)$.

The resulting bits b_1, b_2 are communicated to S_1 who then notifies the client of the accept (if both checks pass) or reject decision. Note that we could output a single bit $b_1 \wedge b_2$ to S_1 to indicate success, but it may be beneficial to differentiate between rejection based on the distance and rejection based on the normalization failure. The former may be the result of authentic user authentication failure, while the latter indicates malfeasance by the client.

The registration and authentication protocols in this model are given as Protocol 1, Reg-SH, and Protocol 2, Auth-SH, respectively. In Protocol 1, client C samples a fresh biometric vector B for enrollment, splits it into XOR shares B_1 and B_2, and sends the shares B_1 and B_2 to S_1 and S_2, respectively. The two servers engage in GC evaluation to determine whether or not the received biometric vector B is unit-length normalized, with S_1 serving the role of the garbler and S_2 the role of the evaluator.

Instead of entering B_1 and B_2 as inputs into GC evaluation, the servers utilize m instances of OTB_1^2 to enter $B_1 \oplus B_2$ directly using the first m wires. The boolean operation of OTB allows for the computation of $B_1[i] \oplus B_2[i]$ outside the GC and is realized as follows. With regular OT, S_1 would supply labels ℓ_i^0 and ℓ_i^1, while S_2

Protocol 2 Authentication Auth-SH

Input: C holds biometric $\widehat{B}$, S_1 holds biometric share B_1, S_2 holds biometric share B_2.
Output: S_1 receives bits b_1 and b_2; C receives accept or reject.
Common Input: Computational security parameter κ and threshold t.
Protocol steps:

(1) C generates m-bit random value $\widehat{B}_2 \xleftarrow{R} \{0,1\}^m$, sets $\widehat{B}_1 = \widehat{B}_2 \oplus \widehat{B}$, and sends $\widehat{B}_2$ to S_2 and $\widehat{B}_1$ to S_1. If the received $\widehat{B}_1$ or $\widehat{B}_2$ is not an m-bit string, the corresponding server signals abort.
(2) S_1 generates labels ℓ_i^j for $i \in [1,n]$ and $j \in \{0,1\}$, computes garbled gates $\mathcal{G}_f$ for the over-the-threshold distance computation and normalization check, and sends $\mathcal{G}_f$ to S_2.
(3) S_1 and S_2 engage in $2m$ instances of OTB_1^2 to communicate to S_2 labels $\ell_i^{B_1[i]\oplus B_2[i]}$ and $\ell_{m+i}^{\widehat{B}_1[i]\oplus\widehat{B}_2[i]}$ for $i \in [1,m]$:
   (a) S_1 enters labels $\ell_i^{B_1[i]}$ and $\ell_i^{B_1[i]\oplus 1}$ into OT, S_2 enters bit $B_2[i]$ and learns label $\ell_i^{B_1[i]\oplus B_2[i]} = \ell_i^{B[i]}$.
   (b) S_1 enters labels $\ell_{m+i}^{\widehat{B}_1[i]}$ and $\ell_{m+i}^{\widehat{B}_1[i]\oplus 1}$ into OT, S_2 enters bit $\widehat{B}_2[i]$ and learns label $\ell_{m+i}^{\widehat{B}_1[i]\oplus\widehat{B}_2[i]} = \ell_{m+i}^{\widehat{B}[i]}$.
(4) S_2 evaluates the circuit and sends the computed labels of the output wires $\ell_{n-1}^{b_2}$ and $\ell_n^{b_1}$ to S_1.
(5) If $\ell_{n-1}^{b_2} = \ell_{n-1}^1$ and $\ell_n^{b_1} = \ell_n^1$, S_1 sends accept to C and terminate to S_2.
(6) Otherwise, S_1 sends reject to C and terminate to S_2.

would supply $B_2[i]$ and receive $\ell_i^{B_2[i]}$. In our protocol, S_1 instead supplies labels $\ell_i^{B_1[i]}$ and $\ell_i^{B_1[i]\oplus 1}$. As a result, when S_1's share $B_1[i] = 0$, the labels are supplied as usual. However, when $B_1[i] = 1$, the supplied labels are swapped relative to usual OT operation. The outcome is that the receiver obtains labels representing the XOR of share bits $B_1[i]$ and $B_2[i]$, or $B[i]$. We can use an OT extension in the implementation.

After circuit evaluation, S_2 obtains the output label ℓ_n^b that represents the outcome of the normalization check and indicates whether registration was successful. S_1 interprets the result and communicates the decision to the other parties.

Protocol 2 proceeds similar to Protocol 1. This time, the parties use $2m$ instances of OTB to communicate GC labels corresponding to inputs $B = B_1 \oplus B_2$ and $\widehat{B} = \widehat{B}_1 \oplus \widehat{B}_2$ to S_2. The output wires with indices $n-1$ and n correspond to the decision bits b_2 and b_1, respectively. If both checks succeed, the client obtain the accept decision and otherwise, it learns that the protocol did not succeed.

Our first security result is as follows:

Theorem 1. *The sequence of Protocols 1 and 2 executed by participants $S_1, S_2, C_{\text{auth}}$ is secure in the presence of semi-honest S_1 and S_2 and malicious C_{auth} according to Definition 2, given supplemental functionalities with security guarantees as discussed in Section 2.3.*

Theorem 2. *The sequence of Protocols 1 and 2, where Protocol 1 is executed by participants $S_1, S_2, C_{\text{auth}}$ and Protocol 2 is executed by participants S_1, S_2, C_{imp}, is secure in the presence of semi-honest S_1 and S_2 and malicious C_{auth} or C_{imp} according to Definition 2, given supplemental functionalities with security guarantees as discussed in Section 2.3.*

The proofs can be found in Appendix A.

Functionality $\mathcal{F}_{\text{reg-mal}}$

(1) $\mathcal{F}_{\text{reg-mal}}$ receives input $B \in \{0,1\}^m$, c, and v from C.
(2) $\mathcal{F}_{\text{reg-mal}}$ samples $r \xleftarrow{R} \{0,1\}^m$ and defines $B_1 = r$ and $B_2 = r \oplus B$.
(3) $\mathcal{F}_{\text{reg-mal}}$ computes $(b_1, b_2) = (\sum_{i=1}^m B[i]^2 \stackrel{?}{=} 1, \text{com}(B, v) \stackrel{?}{=} c)$.
(4) $\mathcal{F}_{\text{auth-mal}}$ outputs (b_1, b_2) and (B_1, c) to S_1 and (B_2, v) to S_2.
(5) In addition, if $(b_1, b_2) = (1,1)$, $\mathcal{F}_{\text{reg-mal}}$ outputs accept to C and S_2; otherwise, it outputs reject to C and S_2.

Figure 3: Ideal registration functionality with malicious and colluding S_2 and C.

3.2 Malicious and colluding C and S_2, semi-honest S_1

We now consider a stronger threat model in which the helper server S_2 can act maliciously and collude with clients C.

When S_2 is not guaranteed to follow the prescribed behavior, it can deviate from the prescribed computation during a protocol execution, but also modify the biometric share B_2 that it receives as part of registration when entering it in the authentication protocol. For that reason, we need to be able to detect this kind of misbehavior in addition to detecting client's misbehavior when it does not use a normalized biometric. Deviations from the prescribed behavior during the protocol execution can be addressed by employing techniques resilient to malicious behavior, while changes to B_2 between protocol executions require a new solution.

Our solution to this problem is to modify the registration phase to enable S_1 to learn a function of S_2's share B_2, which is later used during the authentication to verify that the share that S_2 inputs matches S_1's verification token. We use a commitment scheme for this purpose: the client is instructed to compute a commitment c to its enrollment biometric B and the commitment c is given to S_1. The binding property of the commitment ensures that it is not feasible for S_2 (or S_2 in collusion with C) to later enter a different biometric $B' \neq B$ that matches commitment c. The random choices v used in producing commitment $c = \text{com}(B, v)$ cannot be disclosed to S_1 because they permit the opening of the commitment (and thus disclosure of B) and for that reason, v is known only to S_2. Note that commitments are used an unconventional way in a three-party setting, but their properties allow us to achieve security in a multi-phase execution.

The ideal functionality for registration in this stronger security model is given in Figure 3. In addition to producing shares B_1 and B_2 of enrollment biometric B, the computation includes two checks:

(1) normalization check for B: $b_1 = (\sum_{i=1}^m (B[i])^2 \stackrel{?}{=} 1)$
(2) check that commitment c matches biometric B: $b_2 = (\text{com}(B, v) \stackrel{?}{=} c)$.

Functionality $\mathcal{F}_{\text{auth-mal}}$

(1) $\mathcal{F}_{\text{auth-mal}}$ receives $B_1 \in \{0,1\}^m$ and c from S_1, $B_2 \in \{0,1\}^m$ and v from S_2, and $\widehat{B} \in \{0,1\}^m$ from C.
(2) $\mathcal{F}_{\text{auth-mal}}$ computes $(b_1, b_2, b_3) = (\text{dist}(B_1 \oplus B_2, \widehat{B}) \overset{?}{\sim} t, \sum_{i=1}^{m} (\widehat{B}[i])^2 \overset{?}{=} 1, \text{com}(B_1 \oplus B_2, v) \overset{?}{=} c)$.
(3) $\mathcal{F}_{\text{auth-mal}}$ outputs (b_1, b_2, b_3) to S_1.
(4) If $(b_1, b_2, b_3) = (1, 1, 1)$, then $\mathcal{F}_{\text{auth-mal}}$ outputs accept to C and terminate to S_2.
(5) Otherwise if $b_1 = 0$ or $b_2 = 0$, then $\mathcal{F}_{\text{auth-mal}}$ outputs reject to C and terminate to S_2.
(6) Otherwise $\mathcal{F}_{\text{auth-mal}}$ signals abort.

Figure 4: Ideal authentication functionality with malicious and colluding S_2 and C.

If registration is successful, S_1 obtains and stores B_1 and c, while S_2 obtains and stores B_2 and v.

At authentication time, the servers contribute their shares of B and $\widehat{B}$ as before, but also the remaining values (c and v) that they received at registration time. This time the authentication functionality computes three checks:

(1) comparison of the distance between enrollment and authentication biometrics B and $\widehat{B}$ to threshold t: $b_2 = (\text{dist}(B_1 \oplus B_2, \widehat{B}) \overset{?}{\sim} t)$
(2) normalization check for $\widehat{B}$: $b_1 = (\sum_{i=1}^{m} (\widehat{B}[i])^2 \overset{?}{=} 1)$
(3) check that commitment c matches submitted biometric B: $b_3 = (\text{com}(B_1 \oplus B_2, v) \overset{?}{=} c)$.

S_1 receives these three bits, which allows it to determine the reason for failure (and address it outside the protocol). If at least one bit is 0, authentication fails. Figure 4 specifies the ideal functionality.

As can be seen from the figure, the ideal functionality is written to differentiate between two authentication failure modes: communicating a reject decision to the client and sending an abort signal. The reason is that when the last check fails ($b_3 = 0$), we know that the failure is due to S_2's misbehavior and the client receives a message that the operation did not go through (as opposed to successfully finished with a negative result). It is also possible for other checks to fail due to S_2's misbehavior, but they can also be a result of the client submitting a biometric which is not normalized or not within the desired distance from the enrollment biometric.

The registration protocol for this setting is called Reg-MAL and is given as Protocol 3. It proceeds by the client generating shares and a commitment, and the servers verifying that they received consistent values and properly normalized input. Similar to the normalization check, the commitment check takes place within the garbled circuit. For concreteness, let $|v| = \kappa_1$, $|c| = \kappa_2$, and the inputs being entered into the GC evaluation as B, v, and c. Secret-shared B is entered into GC evaluation via OTB as before, while v is entered using conventional OT. We have to resort to a maliciously secure variant of OT to guarantee correct execution. Recall that GC evaluation itself is resilient to malicious behavior. At the end of GC evaluation, S_2 obtains the output labels $\ell_{n-1}^{b_2}$ and $\ell_n^{b_1}$ that it communicates to S_1. Note that S_2 can tamper with them prior to sending. If the received labels correspond to bits 1, S_1 announces

Protocol 3 Registration Reg-MAL

Input: C holds biometric B.
Output: S_1 receives bits b_1 and b_2, biometric share B_1, and verification token c; S_2 receives accept or reject, biometric share B_2, and verification supplement v; C receives accept or reject.
Common Input: Computational security parameter κ and statistical security parameter ρ.
Protocol steps:

(1) C generates m-bit random value $r \overset{R}{\leftarrow} \{0,1\}^m$, sets $B_1 = r$ and $B_2 = B_1 \oplus B$, and computes $c = \text{com}(B, v)$ using freshly generated randomness v.
(2) C securely communicates (B_1, c) to S_1 and (B_2, v) to S_2. If any communicated value is malformed, the corresponding server signals abort.
(3) S_1 generates labels ℓ_i^j and garbled gates $\mathcal{G}_f$ for the normalization and commitment checks and sends $\mathcal{G}_f$ to S_2.
(4) S_1 and S_2 engage in m instances of maliciously secure OTB to communicate to S_2 labels $\ell_i^{B_1[i] \oplus B_2[i]}$ for $i \in [1, m]$ as in prior protocols and κ_1 instances of conventional maliciously secure OT to communicate to S_2 labels $\ell_{m+i}^{v[i]}$ for $i \in [1, \kappa_1]$. S_1 also sends labels $\ell_{m+\kappa_1+i}^{c[i]}$ for $i \in [1, \kappa_2]$ to S_2.
(5) S_2 evaluates the circuit and communicates the output labels $\ell_{n-1}^{b_2}$ and $\ell_n^{b_1}$ to S_1.
(6) S_1 performs the following:
   (a) If $\ell_{n-1}^{b_2} = \ell_{n-1}^1$ and $\ell_n^{b_1} = \ell_n^1$, S_1 broadcasts accept. S_1 stores (B_1, c) and S_2 stores (B_2, v).
   (b) Otherwise, S_1 sends reject to all parties.

successful completion. If at least one of the labels is invalid, S_1 aborts. Otherwise, it sends a reject signal.

Authentication is termed Auth-MAL and is given as Protocol 4. The changes to the previous authentication protocol include: (i) the addition of commitment inputs (c, v), (ii) the use of OT for entering v, (iii) changes to the circuit to perform commitment verification, (iv) the use of maliciously secure OT, and (v) different handling of the results of function evaluation by S_1. We assume that the circuit wires are allocated to the inputs in the following order: B (m bits), $\widehat{B}$ (m bits), v (κ_1 bits), and c (κ_2 bits). As before, the circuit size is denoted by n, while the output wires this time are $n-2$, $n-1$, n.

Authentication is successful when all output bits are 1 (i.e., all three checks pass). Any malformed output labels and the failure of the commitment check point to S_2's misbehavior and result in abort, while failures of the normalization check and a large difference between B and $\widehat{B}$ can be due to C or S_2 and result in reject.

Theorem 3. *The sequence of Protocols 3 and 4 executed by participants $S_1, S_2, C_{\text{auth}}$ is secure in the presence of semi-honest S_1, and malicious and colluding S_2 and C_{auth}, according to Definition 2, given supplemental functionalities with security guarantees as discussed in Section 2.3.*

Theorem 4. *The sequence of Protocols 3 and 4, where Protocol 3 is executed by participants $S_1, S_2, C_{\text{auth}}$ and Protocol 4 is executed by participants S_1, S_2, C_{imp}, is secure in the presence of semi-honest S_1,*

Protocol 4 Authentication Auth-MAL

Input: C holds biometric $\widehat{B}$, S_1 holds biometric share B_1 and verification token c; S_2 holds biometric share B_2 and verification supplement v.
Output: S_1 receives bits b_1, b_2, and b_3; C receives accept or reject.
Common Input: Computational security parameter κ, statistical security parameter ρ, and threshold t.
Protocol steps:

(1) C generates m-bit random value $\widehat{B}_2 \xleftarrow{R} \{0,1\}^m$, sets $\widehat{B}_1 = \widehat{B}_2 \oplus \widehat{B}$, and sends $\widehat{B}_1$ to S_1 and $\widehat{B}_2$ to S_2. If any received value is malformed, then the corresponding server signals abort.
(2) S_1 generates labels ℓ_i^j, computes garbled gates $\mathcal{G}_f$ for the over-the-threshold distance computation, normalization check, and commitment verification, and sends $\mathcal{G}_f$ to S_2.
(3) S_1 and S_2 engage in $2m$ instances of maliciously secure OTB to communicate to S_2 labels $\ell_i^{B_1[i]\oplus B_2[i]}$ and $\ell_{m+i}^{\widehat{B}_1[i]\oplus\widehat{B}_2[i]}$ for $i \in [1,m]$ and κ_1 instances of conventional maliciously secure OT to communicate to S_2 labels $\ell_{2m+i}^{v[i]}$ for $i \in [1,\kappa_1]$. S_1 also sends labels $\ell_{2m+\kappa_1+i}^{c[i]}$ for $i \in [1,\kappa_2]$ to S_2.
(4) S_2 evaluates the circuit and sends the computed output labels $\ell_{n-2}^{b_3}$, $\ell_{n-1}^{b_2}$, and $\ell_n^{b_1}$ to S_1.
(5) S_1 performs the following:
   (a) If $\ell_{n-2}^{b_3} = \ell_{n-2}^1$, $\ell_{n-1}^{b_2} = \ell_{n-1}^1$, and $\ell_n^{b_1} = \ell_n^1$, S_1 sends accept to C and terminate to S_2.
   (b) If $\ell_{n-1}^{b_2} = \ell_{n-1}^0$ or $\ell_n^{b_1} = \ell_n^0$, S_1 ends reject to C and terminate to S_2.
   (c) Otherwise, S_1 signals abort to C and S_2.

and malicious and colluding S_2 and (C_{auth} or C_{imp}), according to Definition 2, given supplemental functionalities with security guarantees as discussed in Section 2.3.

The proofs can be found in Appendix A.

4 IMPLEMENTATION AND EVALUATION

4.1 Working with Compressed DeepPrint Representation

The use of GCs permits implementing any desired functionality and we realize DeepPrint's matching using compressed representation to lower the cost of the computation. Recall that the main benefit of compressed DeepPrint representation is to lower its storage cost, as the value is uncompressed during the matching. However, in the context of this work, shorter bitlength representation and the use of integer instead of floating-point values can aid efficiency of the computation itself. Thus, we would like to compute as much as possible using the compressed form in a manner which is not lossy with respect to this compression heuristic.

To this end, suppose that we are comparing two DeepPrint representations X and Y consisting of w(=192) elements. Recall that the compressed representation of X uses h_X, ℓ_X together with 8-bit integers $\widehat{X}[i]$ defined in equation 1. We also define $\Delta_X = (h_X - \ell_X)/255 > 0$, represent a compressed biometric as $\widehat{X} \stackrel{\text{def}}{=} (\Delta_X, \ell_X, \{\widehat{X}[i]\}_i)$, and define its decompressed 32-bit (single precision) floating point biometric as $\widetilde{X} = \{\widetilde{X}[i]\}_i$, where $\widetilde{X}[i] \stackrel{\text{def}}{=} \widehat{X}[i]\Delta_X + \ell_X$.

When using cosine similarity comparing normalized vectors which have been compressed and subsequently decompressed, as is done in [17], it suffices to compare the dot product $\widetilde{X} \cdot \widetilde{Y}$ against a threshold value. With this in mind, we have

$$\begin{aligned}
\widetilde{X} \cdot \widetilde{Y} &= \sum_{i=1}^{w} \widetilde{X}[i]\widetilde{Y}[i] = \sum_{i=1}^{w} (\widehat{X}[i]\Delta_X + \ell_X)(\widehat{X}[i]\Delta_Y + \ell_Y) \\
&= \sum_{i=1}^{m} (\widehat{X}[i]\widehat{Y}[i]\Delta_X\Delta_Y + \ell_X\widehat{Y}[i]\Delta_Y + \ell_Y\widehat{X}[i]\Delta_X + \ell_X\ell_Y) \\
&= \left(\Delta_X\Delta_Y \sum_{i=1}^{w} \widehat{X}[i]\widehat{Y}[i]\right) + \left(\ell_X\Delta_Y \sum_{i=1}^{w} \widehat{Y}[i]\right) \\
&+ \left(\ell_Y\Delta_X \sum_{i=1}^{w} \widehat{X}[i]\right) + w\ell_X\ell_Y
\end{aligned}$$

In the final line of this equation, there are w 8-bit multiplications outputting 16-bit values, which are much cheaper than floating-point or even 32-bit integer operations. The summations then require $8 + \lceil \log_2(w) \rceil$ bits to represent the $\sum_{i=1}^{w} \widehat{X}[i]$ and $\sum_{i=1}^{w} \widehat{Y}[i]$ terms, and $16 + \lceil \log_2(w) \rceil$ bits for the $\sum_{i=1}^{w} \widehat{X}[i]\widehat{Y}[i]$ term. Once the computation is performed on short values, we convert the sums to floating-point representation and compute the remaining operations using regular floating-point arithmetic. This adds 3 conversions, 8 single-precision floating-point multiplications and 3 32-bit floating-point additions.

Conversion to a floating-point value involves locating the index of the most significant non-zero bit, shifting the mantissa by this value, and adjusting the exponent by that value as well. Other operations such as floating-point addition involve shifting by an oblivious value as well.

Euclidean distance can be computed over compressed values as

$$\begin{aligned}
\sum_{i=1}^{w} (\widetilde{x}_i - \widetilde{y}_i)^2 &= \sum_{i=1}^{w} (\widehat{X}[i]\Delta_X - \widehat{Y}[i]\Delta_Y + \ell_X - \ell_Y)^2 \\
&= \left(\Delta_X^2 \sum_{i=1}^{w} \widehat{X}[i]^2\right) + \left(\Delta_Y^2 \sum_{i=1}^{w} \widehat{Y}[i]^2\right) \\
&- \left(2\Delta_X\Delta_Y \sum_{i=1}^{w} \widehat{X}[i]\widehat{Y}[i]\right) + \left(2(\ell_X - \ell_Y)\Delta_X \sum_{i=1}^{w} \widehat{X}[i]\right) \\
&- \left(2(\ell_X - \ell_Y)\Delta_Y \sum_{i=1}^{w} \widehat{Y}[i]\right) + w(\ell_X - \ell_Y)^2
\end{aligned}$$

which shows an increase in the number of short integer, floating-point, and integer to floating-point conversion operations.

4.2 Circuit Optimizations

We employ a number of optimizations within our circuits in an effort to make our implementation as efficient as possible.

The operations described above need to be implemented using garbled circuits. Our GC implementation was built to maximize efficiency starting from the lowest levels. For example, the initial codebase for JustGarble [9] does not have provisions for efficiently

incorporating constant publicly known values. We build this into the code systematically, giving a special designation to any wire carrying a public constant. This allows us to employ the logic used in OTB to eliminate unnecessary gate execution. Specifically, if at least one input to any AND, OR, or XOR gate is a public constant, then in almost all cases no gate execution is needed. For example, for an AND gate:

- if one input is 0, then 0 is passed though;
- If one input is 1, then the other value is passed though;
- If the inputs are the same, meaning that they come from the same wire or have the same constant value, then the input is passed though.

Furthermore, if both inputs into a gate are constant, then the output will also be constant. And if constant inputs on average have a similar number of 0s and 1s, nearly half of the outputs will on average be constant. This optimization can, for example, cut approximately in half the number of gates needed to evaluate multiplication when compared to the built-in multiplication in JustGarble (where intermediate values are represented using $2n$ bits, while after shifting an n-bit argument, half of the bits are 0).

We layer higher-level circuit optimizations on top of this foundation. In order to discuss this further, we define additional notation. We use double-square brackets to denote private variables represented as bit vectors of some length; e.g., $[\![a]\!]$. We refer to a subset of such vectors using array notation with 0-indexing, so that $[\![a]\!][j-1..0]$ refers to the j least significant bits of $[\![a]\!]$. Unless otherwise noted, n is refers to the bitlength of the primary (integer) input into a function. We also denote bit strings using notation such as 0^i1^j, which in this example represents i 0s followed by j 1s, where we assume the most significant bit is on the left. We may combine these notations with concatenation, so that, e.g., $0^i[\![a]\!][n-1..n-k]||1^j$ refers to i 0s, followed by the k most significant bits of $[\![a]\!]$, followed by j 1s, reading from most significant bit to least.

Most of the cases in which we prepend or append a vector with 0s is equivalent to shifting an private bit vector by a publicly known value, which by itself does not incur any gates. This stands in contrast to the Obliv_Shift function we will discuss later, which shifts a private value by a private (oblivious) amount.

One circuit-level optimization we discuss is an integer squaring routine, which we also extend to squaring floating-point values. At a high level, it operates by splitting an n-bit input a into two parts of equal length a_{hi} and a_{lo}, where $a = 2^{n/2} \cdot a_{hi} + a_{lo}$. Then it recursively computes the squares of each of these halves, along with their product (the latter uses standard textbook multiplication). The algorithm recurses until the input length is no greater than the stopping bitlength parameter, denoted by *stop*. Note that we generally find a $stop = 4$ to be optimal in most situations and use it for all computation in this work. The construction is given as Algorithm 1.

The algorithm runs in $O(n^2)$ time and thus is not asymptotically better than textbook multiplication that performs a series of shifts and additions. It does, however, benefit from more favorable constants and tends to half as many gates as that of textbook multiplication. Let an "operation" to be one AND gate plus a full 1-bit adder (which is realized in JustGarble with 1 AND and 4 XOR gates) and j be the depth of recursion in the Square algorithm. Then line 9 in Algorithm 1 incurs $\sum_{i=1}^{j} 2^{i-1}(2^{-i}n)^2 = (2^{-1} - 2^{-j-1})n^2$ operations over all recursive instances, line 2 incurs $2^j(n2^{-j})^2 = 2^{-j}n^2$ operations over all instances, and line 13 incurs $\sum_{i=1}^{j} 3 \cdot 2^{i-1}(2^{-i}n) = 3nj/2$ operations over all instances. In total, we have $(2^{-1} + 2^{-j-1})n^2 + (3 \cdot 2^{-1} \cdot j)n$ operations, which asymptotically tends towards $n^2/2$ for large n, whereas textbook multiplication performs exactly n^2 operations. Also note that the summation on line 13 benefits from the systematic handling of addition with 0, which implies the factor of 3 instead of 4 in the above analysis. This algorithm is not to be confused with the recursive Karatsuba multiplication [24] of lower complexity $O(n^{\log_2 3})$. We find that Karatsuba multiplication becomes more efficient for inputs above about 32 bits, which is larger than the integers we are multiplying. Thus, Karatsuba multiplication was not used in our implementation.

Algorithm 1 $[\![result]\!]$ = Square($[\![a]\!]$, n, *stop*)

1: **if** $n \leq stop$ **then**
2: **return** $[\![a]\!] \cdot [\![a]\!]$
3: **end if**
4: $[\![a_{hi}]\!] = [\![a]\!][n-1..n/2]$
5: $[\![a_{lo}]\!] = [\![a]\!][n/2-1..0]$
6: $n \gg= 1$
7: $[\![a_{hi}^2]\!]$ = Square($[\![a_{hi}]\!]$, n, *stop*)
8: $[\![a_{lo}^2]\!]$ = Square($[\![a_{lo}]\!]$, n, *stop*)
9: $[\![a_{mid}]\!] = [\![a_{hi}]\!] \cdot [\![a_{lo}]\!]$
10: $[\![a_{hi}^2]\!] = [\![a_{hi}^2]\!]||0^{2 \cdot n}$
11: $[\![a_{mid}]\!] = 0^{n-1}||[\![a_{mid}]\!]||0^{n+1}$
12: $[\![a_{lo}^2]\!] = 0^{2 \cdot n}||[\![a_{lo}^2]\!]$
13: $[\![result]\!] = [\![a_{hi}^2]\!] + [\![a_{mid}]\!] + [\![a_{lo}^2]\!]$
14: **return** $[\![result]\!]$

In our floating-point squaring routine, squaring occurs on 24-bit mantissas. The recursive squaring algorithm alone reduces the number of gates from 3456 in standard multiplication to 2592 gates (one third of which are AND gates). However, both our squaring routine and textbook multiplication perform even better. This is because some computation that operates on two identical values (e.g., that on line 2 of Algorithm 1) can benefit from the "inputs are the same" gate-level optimization that eliminates some gates. This permits conventional multiplication of 24-bit integers using 3266 gates, while other squaring algorithm uses 1931 gates.

We employ a number of other optimizations that allowed us to generate very efficient circuits, particularly for floating-point operations. That is, we are aware of only one other work [32] that built garbled circuits for floating-point operations, and our circuits compare very favorably. Our single-precision floating-point addition uses 2030 gates vs. 7052 gates in [32]; our multiplication has 3690 gates vs. 7701 gates in [32], and comparison is efficient at 300 gates (not tested in [32]). We also employ optimized circuits for summation of multiple values, over both integer and floating-point values. We note that there can be differences in the treatment of exceptions (e.g., we treat infinity as non-a-number NaN which

Algorithm 2 $\langle [\![idx]\!], [\![nz]\!] \rangle = \text{MSNZB}([\![a]\!], n)$

1: $[\![p]\!] = 0 || \text{Prefix\_Or}([\![a]\!])$
2: $[\![mask]\!] = (0||[\![p]\!]) \oplus ([\![p]\!]||0)$
3: $[\![mask]\!] = [\![mask]\!][n..1]$
4: $[\![nz]\!] = [\![p]\!][0]$
5: $[\![idx]\!] = 0$
6: **for** $i = 0$ to $n-1$ **do**
7: $\quad [\![tmpidx]\!] = i \wedge [\![mask]\!][i]$
8: $\quad [\![idx]\!] \oplus = [\![tmpidx]\!]$
9: **end for**
10: **return** $\langle [\![idx]\!], [\![nz]\!] \rangle$

improves performance), but we expect that even with full IEEE 754 standard treatment, our circuits will compare favorably.

We next describe two optimized circuits which are needed to convert a private integer to a floating point value. The first is determining the most significant non-zero bit (MSNZB) of an integer. This is used for a number of operations including comparisons (both integer and floating-point) and is realized in a two-stage Algorithm 2. The first stage produces a mask, in which for any given input, the output stores a 1 in the MSNZB position (if any) and 0 everywhere else. The second stage converts the mask into a base 2 representation of the MSNZB position. This is used, e.g., during conversion of an integer to a floating-point value, since the exponent depends on the index of the MSNZB. The algorithm produces two values: the index of the MSNZB *idx* and a flag *nz* indicating whether there were no non-zero bits.

The prefix OR operation on line 1 of Algorithm 2 computes an array of bits, where the ith bit is set to the OR of bits $a[i]$ through $a[n-1]$. This means that the most significant bits of the output will be 0 until we come across the first non-zero bit, after which all other bits will be set to 1. We obtain that after calling line 1, p will be of the form $0^{n+1-j}1^j$, where the MSNZB is at index j. Lines 2–3 convert the *mask* value to be of the form where only the bit in the MSNZB position is set, i.e., it will be of the form $0^{n-j}10^{j-1}$ (this computation coincides with the inverse prefix XOR computation). Prepending and appending extra 0 bits on lines 1–3 is used for simplifying the algorithm.

This algorithm can also be used to demonstrate the impact of efficiently incorporating constant values, which results in many operations in the loop on lines 6–9 to not incur gates. That is, indices are constant, which means that for any set bit of index i the other value is passed through to the corresponding bit in *tmpidx*. And for any unset bit in i, a constant zero value is fed into the corresponding bit in *tmpidx*. The latter case also results in passing the value through during the computation on line 8.

Another interesting functionality is Algorithm 3 that shifts a private integer by a private (oblivious) amount. This algorithm works by iterating through each bit of the binary representation of the amount to shift and branching into two cases: one which shifts the input by the amount represented by that bit position and one which doesn't. Then one of those options is selected based on the actual value of the bit on line 10. As always, conditional selection can be written as $[\![a\_sh]\!] = ([\![a\_sh]\!] \oplus [\![prev]\!]) \wedge [\![obl\_sh]\!][i] \oplus [\![prev]\!]$, which lowers the number of AND gates by a factor of 2.

Algorithm 3 $[\![a\_sh]\!] = \text{Obliv\_Shift}([\![a]\!], n, dir, maxsh, [\![obl\_sh]\!])$

1: $[\![a\_sh]\!] = [\![a]\!]$
2: $cur\_sh = 1$
3: **for** $i = 0$ to $\lfloor \log_2(maxsh) \rfloor$ **do**
4: $\quad [\![prev]\!] = [\![a\_sh]\!]$
5: $\quad$ **if** $dir = left$ **then**
6: $\quad\quad [\![a\_sh]\!] = [\![prev]\!][(n-1-cur\_sh)..0]||0^{cur\_sh}$
7: $\quad$ **else**
8: $\quad\quad [\![a\_sh]\!] = 0^{cur\_sh}||[\![prev]\!][(n-1)..cur\_sh]$
9: $\quad$ **end if**
10: $\quad [\![a\_sh]\!] = ([\![obl\_sh]\!][i] \wedge [\![a\_sh]\!]) \oplus (\neg[\![obl\_sh]\!][i] \wedge [\![prev]\!])$
11: $\quad cur\_sh *= 2$
12: **end for**
13: **return** $[\![a\_sh]\!]$

Algorithm 4 $\langle [\![mant]\!], [\![exp]\!], [\![sign]\!] \rangle = \text{Integer\_to\_Float}([\![a]\!], n)$

1: $\langle [\![nz]\!], [\![idx]\!] \rangle = \text{MSNZB}([\![a]\!], n)$
2: $[\![exp]\!] = [\![idx]\!] + 127$
3: $[\![obl\_sh]\!] = n + 1 - [\![idx]\!]$
4: **if** $n \geq 23$ **then**
5: $\quad [\![a\_sh]\!] = \text{Obliv\_Shift}([\![a]\!], n, left, n, [\![obl\_sh]\!])$
6: $\quad [\![mant]\!] = [\![a\_sh]\!][n-1..n-23]$
7: **else**
8: $\quad [\![a\_sh]\!] = [\![a]\!][n-1..0]||0^{23-n}$
9: $\quad [\![mant]\!] = \text{Obliv\_Shift}([\![a\_sh]\!], n, left, n, [\![obl\_sh]\!])$
10: **end if**
11: $[\![exp]\!] \wedge = [\![nz]\!]$
12: **return** $\langle [\![mant]\!], [\![exp]\!], 0 \rangle$

The above two algorithms are useful for conversion from integer to floating-point representation, which we provide as Algorithm 4. We compute the position of the MSNZB of the input and use the bits starting from that position to form the significand, and set the exponent also based on the position of the MSNZB. In IEEE 754 specification, the exponent is in the range $[-126, 127]$ represented as unsigned, and thus we adjust it by adding 127. Formsing the significand differs based on whether the input's bitlength n is smaller than the single-precision signficand's bitlength (23) or not. That is, if $n < 23$, the input is shifted left and right-padded with 0s to form a 23-bit vector prior to oblivious shift to set the MSNZB in the MSB position. And if $n \geq 23$, the oblivious shift occurs first, followed by truncation of the $n-23$ least significant bits. Our experiments in Section 4.3 use $n = 8$. This procedure is specific to unsigned integer inputs and thus the sign in the result is always 0.

4.3 Experimental Evaluation

Our implementation uses the GC instantiation from [9] with the free-XOR and row reduction optimizations, along with those discussed above. The OT extension is from [5] for the protocols with semi-honest S_2, and from [6] for the protocols with malicious S_2. Both of them build on an optimized version of the semi-honest protocol from [22] to generate the base OTs. Commitments are formed using SHA-256 with $\kappa_2 = 256$ and $|v| = \kappa_1 = 128$ random bits included as entropy supplement. $m = 1600$ and reported performance

Protocol	Dist. metric	Circuit gates	
		Comm.	Local
Auth-SH	CS	63,017	117,327
	ED	82,862	152,834
Auth-MAL	CS	176,669	489,109
	ED	196,514	524,616

Table 1: Circuit complexity, in gates and bytes

Protocol	Dist. metric	LAN online time				Online comm.
		GCE	OT	Other	Total	
Auth-SH	CS	12.2	12.7	2.20	27.1	216KB
	ED	15.4	12.8	2.23	30.5	
Auth-MAL	CS	32.2	17.6	3.05	52.9	313KB
	ED	35.1	17.7	3.15	56.0	

Table 2: Performance of LAN authentication protocol, online phase; runtime is in ms.

Protocol	Dist. metric	Internet online time				Online comm.
		GCE	OT	Other	Total	
Auth-SH	CS	13.4	13.3	48.9	75.6	216KB
	ED	16.9	13.5	49.1	79.5	
Auth-MAL	CS	33.0	18.0	49.9	101	313KB
	ED	36.3	17.9	49.8	104	

Table 3: Performance of Internet authentication protocol, online phase; runtime is in ms.

Protocol	Dist. metric	Offline time			Offline comm.
		Garble	Send	Total	
Auth-SH	CS	24.9	60.2	92.7	8.26 MB
	ED	32.6	82.8	125	10.8 MB
Auth-MAL	CS	76.2	259	364	31.8 MB
	ED	83.8	282	397	34.3 MB

Table 4: Performance of authentication protocols, offline phase; runtime is in ms.

is averaged over 100 runs. The implementation is available as open source from https://github.com/applied-crypto-lab/biom-auth.

The the following machines were used to run the experiments:

- An AMD Ryzen5-3600 6-core processor machine operating at 3.6 GHz, running openSUSE Leap 15.3 on the GNU/Linux kernel 5.3.18-150300.59.101-default.
- Identical computers with Intel Xeon E5-2620v4 8-core processors operating at 2.1GHz, running Ubuntu 20.04.3 LTS on the GNU/Linux kernel 5.4.0-131-generic x86_64.

All communication used TCP sockets with the following setup:

- Each party on a different Xeon machine on a LAN.
- The two servers on Xeon machines on a LAN, with the client on the Ryzen machine connecting via VPN from the internet.

Network latency and throughput were measured by transmitting buffers of size 4^i bytes for $i \in [0, 12]$ bidirectionally. Round trip time (RTT) is taken to be the average of transmitting ≤ 256 bytes (representing one packet). Throughput is taken to be $((2 \cdot 8 \cdot \text{bufsize}) - \text{latency})/\text{time(sec)}$ as a buffer of size bufsize is sent twice in this round trip test. We obtain RTT of 0.345 ms and throughput of 946 Mb/sec for Xeon LAN and RTT of 45.9 ms and throughput of 20.4 Mb/sec for Ryzen to Xeon over internet. We use encrypted channels for sensitive information such as biometric shares, GC labels, and protocol outcomes (while garbled tables and OTB communication are not encrypted).

Tables 1, 2, 3, and 4 report performance of authentication protocols. The online time and communication correspond to all work with the exception of garbling and transmitting the garbled table and labels from S_1 to S_2, which can be precomputed and constitutes offline work. Note that online communication is independent of the distance metric, while offline runtime is independent of the network setup (as the connection between the servers does not change). The GC size is sub-divided into gates that involve communication (AND and OR) and those that do not (XOR and NOT). The category "Other" of online time includes communication time, other local computation time, and down time. The (online) communication time is relatively insignificant for LAN tests but dominates the mixed internet test times.

The client computation time is not included, but it is minimal, showing that the solution is well suited for constrained devices. In particular, client's computation took on overage 0.17 ms with additional 0.05 ms for data transmission to the socket, after which the client awaits a response. In addition, the client only transmits 400B (and receives 1B), while the remaining communication is between the servers.

One can see from the tables that cosine similarity is slightly more efficient than squared Euclidean distance computation. While some of the optimizations such as the squaring algorithm and reusing the computed sum of compressed input squares in the normalization check can be used only by the Euclidean distance computation, this is not enough to offset the difference.

We can can compare performance of our solutions with that of other constructions that treat biometric-based authentication and consider at least the client to be malicious [2, 16, 20, 21, 34]. All of them require the clients to store keys on their devices and all with the exception of [20] consider only semi-honest servers (and thus are closer to our first threat model). Among these, [34] has performance on the order of seconds or larger and is not suitable for real-time authentication. [21] takes about a second for a computation that leaks the distance to the server in the semi-honest server model. [16] does not provide sufficient information to determine the time, but it is lower-bounded by several hundred ms. With [2], authentication takes 71ms on 128-byte vectors and 102ms on 256-byte vectors without taking communication into account, both of which are higher than our 192-element times are. This protocol does not leak the distance to the server, but does not achieve security (the client notifies the server of authentication outcome). Lastly, in [20] authentication (client and server work) takes over 150ms with much shorter 64-byte templates and over 175ms with 128-byte templates without taking network communication into

account and disclosing the distance to the server. Once again, this is slower than performance of all of our protocols.

5 CONCLUSIONS

In this work, we treat the topic of privacy-preserving biometric-based authentication that permits users to authenticate with biometric data in a such a way that users do not have to maintain any additional secrets and the authentication server does not learn information about user biometrics. We build solutions using a number of cryptographic techniques such as garbled circuit evaluation, a new variant of oblivious transfer, and a commitment scheme that rely on a helper server. An interesting aspect of our work is that the standard security definitions adopted in secure multi-party computation literature were not sufficient to demonstrate security in our application and we extend them to accommodate computation consisting of multiple phases where the set of participants might change from one phase to another. We consider two different security models, both of which model users as malicious and differ in the assumptions on the servers. We formally prove all of our constructions to be secure in the respective models and implement them to demonstrate that they have practical performance.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their helpful feedback. This work was supported in part by NSF grants 1822190 and 2213057 and a Google Faculty Award. Any opinions, findings, and conclusions expressed in this publication are those of the authors and do not necessarily reflect the views of the funding sources.

REFERENCES

[1] S. Agrawal, S. Badrinarayanan, P. Mohassel, P. Mukherjee, and S. Patranabis. BETA: Biometric-enabled threshold authentication. In *Public-Key Cryptography (PKC)*, pages 290–318, 2021.

[2] S. Agrawal, S. Badrinarayanan, P. Mukherjee, and P. Rindal. Game-Set-MATCH: Using mobile devices for seamless external-facing biometric matching. In *ACM Conference on Computer and Communications Security*, pages 1351–1370, 2020.

[3] S. Agrawal, P. Miao, P. Mohassel, and P. Mukherjee. Pasta: PASsword-based Threshold Authentication. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, page 2042–2059, 2018.

[4] M. Aliasgari, M. Blanton, and F. Bayatbabolghani. Secure computation of hidden Markov models and secure floating-point arithmetic in the malicious model. *International Journal of Information Security*, 16:577–601, 2017.

[5] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 535–548, 2013.

[6] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. Cryptology ePrint Archive, Report 2015/061, 2015.

[7] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Donida Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti, and A. Piva. Privacy-preserving fingercode authentication. In *ACM Workshop on Multimedia and Security*, pages 231–240, 2010.

[8] F. Bayatbabolghani, M. Blanton, M. Aliasgari, and M. Goodrich. Secure fingerprint alignment and matching protocols. arXiv Report 1702.03379, 2017.

[9] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE S&P*, pages 478–492, 2013.

[10] M. Blanton and M. Aliasgari. On the (non-)reusability of fuzzy sketches and extractors and security in the computational setting. In *International Conference on Security and Cryptography (SECRYPT)*, pages 68–77, 2011.

[11] M. Blanton and M. Aliasgari. Secure outsourced computation of iris matching. *Journal of Computer Security*, 20(2–3):259–305, 2012.

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

[13] M. Blanton and S. Saraph. Oblivious maximum bipartite matching size algorithm with applications to secure fingerprint identification. In *European Symposium on Research in Computer Security (ESORICS)*, pages 384–406, 2015.

[14] V. Boddeti. Secure face matching using fully homomorphic encryption. In *IEEE International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–10, 2018.

[15] F. Catak, S. Yildirim Yayilgan, and M. Abomhara. A privacy-preserving fully homomorphic encryption and parallel computation based biometric data matching. Preprints manuscript 2020070658, 2020.

[16] J. Cheon, H. Chung, M. Kim, and K.-W. Lee. Ghostshell: Secure biometric authentication using integrity-based homomorphic evaluations. IACR Cryptology ePrint Archive Report 2016/484, 2016.

[17] J. Engelsma, K. Cao, and A. Jain. Learning a fixed-length fingerprint representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(6):1981–1997, 2021.

[18] J. Engelsma, A. Jain, and V. Boddeti. HERS: Homomorphically encrypted representation search. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 4(3):349–360, 2022.

[19] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies Symposiym (PETS)*, pages 235–253, 2009.

[20] J. Ernst and A. Mitrokotsa. A framework for UC secure privacy preserving biometric authentication using efficient functional encryption. In *Applied Cryptography and Network Security (ACNS)*, pages 167–196. 2023.

[21] J.-K. Im, S.-Y. Jeon, and M.-K. Lee. Practical privacy-preserving face authentication for smartphones secure against malicious clients. *IEEE Transactions on Information Forensics and Security (TIFS)*, 15:2386–2401, 2020.

[22] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO*, pages 145–161, 2003.

[23] A. Juels and M. Sudan. A fuzzy vault scheme. *Design, Codes and Cryptography*, 38:237–257, 2006.

[24] A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. 1962.

[25] B. Karmakar, N. Koti, A. Patra, S. Patranabis, P. Paul, and D. Ravi. Asterisk: Super-fast MPC with a friend. In *IEEE Symposium on Security and Privacy (S&P)*, pages 127–127, 2024.

[26] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming*, pages 486–498, 2008.

[27] S. Kumar, D. Culler, and R. Popa. MAGE: Nearly zero-cost virtual memory for secure computation. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 367–385, 2021.

[28] Y. J. Lee, K. R. Park, S. J. Lee, K. Bae, and J. Kim. A new method for generating an invariant iris private key based on the fuzzy vault system. *IEEE Transactions on Systems, Man and Cybernetics. Part B, Cybernetics*, 38(5):1302–1313, 2008.

[29] M. Morampudi, M. Prasad, and U. Raju. Privacy-preserving iris authentication using fully homomorphic encryption. *Multimedia Tools and Applications*, 79:19215–19237, 2020.

[30] K. Nandakumar, A. Nagar, and A. Jain. Hardening fingerprint fuzzy vault using password. In *International Conference on Advances in Biometrics (ICB)*, pages 927–937, 2007.

[31] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. SCiFI - a system for secure face identification. In *IEEE Symposium on Security and Privacy*, pages 239–254, 2010.

[32] P. Pullonen and S. Siim. Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations. In *Financial Cryptography and Data Security*, pages 172–183, 2015.

[33] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *International Conference on Information Security and Cryptology (ICISC)*, pages 229–244, 2010.

[34] J. Sedenka, S. Govindarajan, P. Gasti, and K. Balagani. Secure outsourced biometric authentication with performance evaluation on smartphones. *IEEE Transactions on Information Forensics and Security (TIFS)*, 10(2):384–396, 2015.

[35] U. Uludag, S. Pankanti, and A. Jain. Fuzzy vault for fingerprints. In *Audio and Video Based Biometric Person Authentication (AVBPA)*, pages 310–319, 2005.

[36] A. C. Yao. Protocols for secure computations. In *Annual Symposium on Foundations of Computer Science (SFCS)*, pages 160–164, 1982.

[37] H. Zhu, Q. Wei, X. Yang, R. Lu, and H. Li. Efficient and privacy-preserving online fingerprint authentication scheme over outsourced data. *IEEE Transactions on Cloud Computing*, 9(2):576–586, 2018.

A SECURITY PROOFS

Proof of Theorem 1. We need to consider each type of the adversary, namely, malicious C_{auth}, semi-honest S_1, and semi-honest S_2. We first note that C_{auth} is the only party contributing input to the computation and thus it is not possible for $\mathcal{A}_{C_{\mathsf{auth}}}$ – malicious adversary corrupting C_{auth} – to violate security and discover information about other parties' inputs. Thus, we proceed with first

showing security in the presence of semi-honest $\mathcal{A}_{S_1}$ controlling S_1 and build a respective simulator $\mathcal{S}_{S_1}$. Afterwards, we treat the case of semi-honest $\mathcal{A}_{S_2}$ controlling S_2 and build its simulator S_{S_2}.

Adversary $\mathcal{A}_{S_1}$. Simulation of Protocol 1, Reg-SH, proceeds as follows:

(1) $\mathcal{S}_{S_1}$ invokes $\mathcal{F}_{\text{reg-sh}}$ and receives bit b and either B_1 or $\perp$.
(2) If $\mathcal{F}_{\text{reg-sh}}$ output B_1 then $\mathcal{S}_{S_1}$ chooses random B_2 subject to $||B_1 \oplus B_2|| = 1$ and stores both values for use in the authentication phase.
(3) Otherwise, $\mathcal{S}_{S_1}$ samples random $B_1 \xleftarrow{R} \{0,1\}^m$ and chooses random B_2 subject to $||B_1 \oplus B_2|| \neq 1$.
(4) $\mathcal{S}_{S_1}$ sends B_1 to $\mathcal{A}_{S_1}$.
(5) $\mathcal{S}_{S_1}$ receives from $\mathcal{A}_{S_1}$ the garbled gates $\mathcal{G}_f$.
(6) $\mathcal{S}_{S_1}$ and $\mathcal{A}_{S_1}$ engage in m instances of OTB, with $\mathcal{S}_{S_1}$ entering B_2 to simulate S_2's participation. As a result, $\mathcal{S}_{S_1}$ receives labels corresponding to $B_1 \oplus B_2$.
(7) $\mathcal{S}_{S_1}$ evaluates the garbled circuit and obtains the output label corresponding to b and sends it to $\mathcal{A}_{S_1}$.

We now argue indistinguishability between real and ideal execution. First note that S_1's view in Protocol 1 is formed by S_1's local randomness and the following components:

(1) Receiving B_1 from C in Protocol 1 Step 1.
(2) Engaging in OTB with S_2 in Protocol 1 Step 3, with S_2 receiving m labels.
(3) Receiving ℓ_n^b from S_2 in Protocol 1 Step 4.
(4) Using ℓ_n^b to determine the computation outcome.

Because $\mathcal{A}_{S_1}$ is semi-honest, it will not deviate from what is prescribed in Protocol 1. We now examine all components above.

For part (1), $\mathcal{S}_{S_1}$ always sends a random B_1 to $\mathcal{A}_{S_1}$, which has identical distribution to the value used in the protocol. For part (2), the security guarantees of OTB are such $\mathcal{A}_{S_1}$ does not learn any information acting as a sender and there is only a negligible in κ chance that it learns the bits input by $\mathcal{S}_{S_1}$, satisfying the indistinguishability requirement.

For parts (3) and (4), $\mathcal{S}_{S_1}$ uses the output of $\mathcal{F}_{\text{reg-sh}}$ in the simulation steps leading up to this point to ensure $\mathcal{A}_{S_1}$ derives the correct output as part of its view, as follows: If $\mathcal{F}_{\text{reg-sh}}$ reports (accept, B_1), then $\mathcal{S}_{S_1}$ uses this value B_1 in OTB and chooses B_2, which given this B_1, is guaranteed to pass the normalization check. If instead $\mathcal{F}_{\text{reg-sh}}$ reports reject, then $\mathcal{S}_{S_1}$ selects a random B_1 which matches the expected share distribution in real execution and chooses B_2 which is guaranteed to fail the normalization check (and will thus trigger rejection of the enrollment biometric).

We obtain that all components of real and ideal executions are indistinguishable to $\mathcal{A}_{S_1}$ except for vanishing probability in κ.

If $\mathcal{S}_{S_1}$ receives accept in the registration phase, we proceed to simulation of Protocol 2, Auth-SH. Recall that $\mathcal{S}_{S_1}$ retains and reuses the values B_1 and B_2 from the registration phase when it was successful. The simulator works as follows:

(1) $\mathcal{S}_{S_1}$ invokes $\mathcal{F}_{\text{auth-sh}}$ and receives output bits (b_1, b_2).
(2) $\mathcal{S}_{S_1}$ samples random $\widehat{B}_1$ and sends $\widehat{B}_1$ to $\mathcal{A}_{S_1}$.
(3) Using B_1 and B_2 from the registration phase simulation, $\mathcal{S}_{S_1}$ chooses $\widehat{B}_2$, subject to the following two constraints:
   (a) If $b_1 = 0$, then $\widehat{B}_2$ is chosen such that $\text{dist}(B_1 \oplus B_2, \widehat{B}_1 \oplus \widehat{B}_2) \geq t$, and otherwise subject to $\text{dist}(B_1 \oplus B_2, \widehat{B}_1 \oplus \widehat{B}_2) < t$.
   (b) If $b_2 = 0$, then $\widehat{B}_2$ is chosen subject to $||\widehat{B}_1 \oplus \widehat{B}_2|| \neq 1$, and otherwise subject to $||\widehat{B}_1 \oplus \widehat{B}_2|| = 1$.
(4) $\mathcal{S}_{S_1}$ receives from $\mathcal{A}_{S_1}$ the garbled gates $\mathcal{G}_f$.
(5) $\mathcal{A}_{S_1}$ and $\mathcal{S}_{S_1}$ engage in $2m$ instances of OTB, with $\mathcal{S}_{S_1}$ entering the bits of $(B_2, \widehat{B}_2)$. As a result, $\mathcal{S}_{S_1}$ receives labels corresponding to $(B_1 \oplus B_2, \widehat{B})$.
(6) $\mathcal{S}_{S_1}$ evaluates the garbled circuit, obtains the output label corresponding to (b_1, b_2), and sends them to $\mathcal{A}_{S_1}$.

S_1's view in Protocol 2 is formed by its local randomness and the following components:

(1) S_1's share B_1 received in Protocol 1, which is used as an auxiliary input to this protocol.
(2) Receiving $\widehat{B}_1$ from C in Protocol 2 Step 1.
(3) Engaging in OTB with S_2 in Protocol 2 Step 3, with S_2 receiving $2m$ labels.
(4) Receiving $\ell_{n-1}^{b_1}$ and $\ell_n^{b_2}$ from S_2 in Protocol 2 Step 3.
(5) Using $\ell_{n-1}^{b_1}$ and $\ell_n^{b_2}$ to determine the computation outcome.

The value in part (1) is output from registration and was discussed above. For part (2), $\mathcal{S}_{S_1}$ sends a random $\widehat{B}_1$ to $\mathcal{A}_{S_1}$, which has identical distribution to the value used in the protocol. And for part (3), the security guarantees of OTB are again such that $\mathcal{A}_{S_1}$ does not learn any information acting as a sender and there is only a negligible in κ chance that it learns the bits input by $\mathcal{S}_{S_1}$.

For parts (4) and (5), $\mathcal{S}_{S_1}$ again uses the output of $\mathcal{F}_{\text{reg-sh}}$ to ensure $\mathcal{A}_{S_1}$ derives the correct output as part of its view. In particular, the bit b_1 received from $\mathcal{F}_{\text{reg-sh}}$ represents the outcome of the biometric distance check, and b_2 represents the outcome of the normalization check. $\mathcal{S}_{S_1}$ then uses the $\widehat{B}_1$ it generated to choose $\widehat{B}_2$ such that the distance check passes if and only if $b_1 = 1$ and the normalization check passes if and only if $b_2 = 1$ (where failure of either will trigger rejection of the authentication attempt). Thus, again the probability that the simulated view diverges from real execution is negligible in κ.

Adversary $\mathcal{A}_{S_2}$. Next we have adversary $\mathcal{A}_{S_2}$ controlling semi-honest S_2 and construct a corresponding simulator $\mathcal{S}_{S_2}$. Simulation of the registration phase proceeds as follows:

(1) $\mathcal{S}_{S_2}$ invokes $\mathcal{F}_{\text{reg-sh}}$ and receives either (B_2, accept) or ($\perp$, reject).
(2) If $\mathcal{F}_{\text{reg-sh}}$ outputs (B_2, accept), then $\mathcal{S}_{S_2}$ chooses random B_1 subject to $||B_1 \oplus B_2|| = 1$ and stores both values for use in the authentication phase.
(3) Otherwise, $\mathcal{S}_{S_2}$ samples random $B_2 \xleftarrow{R} \{0,1\}^m$, chooses B_1 subject to $||B_1 \oplus B_2|| \neq 1$, and stores both values for use in the authentication phase.
(4) $\mathcal{S}_{S_2}$ sends B_2 to $\mathcal{A}_{S_2}$.
(5) $\mathcal{S}_{S_2}$ constructs a garbled circuit as per Protocol 1 sends the garbled gates $\mathcal{G}_f$ to $\mathcal{A}_{S_2}$.
(6) $\mathcal{S}_{S_2}$ and $\mathcal{A}_{S_2}$ engage in m instances of OTB, with $\mathcal{S}_{S_2}$ entering the bits of B_1. As a result, $\mathcal{A}_{S_2}$ receives labels corresponding to $B_1 \oplus B_2$.
(7) $\mathcal{S}_{S_2}$ receives from $\mathcal{A}_{S_2}$ the output label corresponding to b.

(8) If $\mathcal{F}_{\text{reg-sh}}$ output $\perp$, then $\mathcal{S}_{S_1}$ sends reject to $\mathcal{A}_{S_2}$, otherwise $\mathcal{S}_{S_2}$ sends accept to $\mathcal{A}_{S_2}$.

S_2's view in Protocol 1 is formed by S_2's local randomness and the following components:

(1) Receiving B_2 from C in Protocol 1 Step 1.
(2) Engaging in OTB with S_1 in Protocol 1 Step 3, with S_2 receiving m labels.
(3) Obtaining ℓ_n^b from garbled circuit evaluation in Protocol 1 Step 4.
(4) Receiving the registration decision from S_1.

For part (1), $\mathcal{S}_{S_2}$ always sends a random B_2 to $\mathcal{A}_{S_2}$, which has identical distribution to the value used in the protocol. For part (2), the security guarantees of OTB are such that there is only a negligible in κ chance that $\mathcal{A}_{S_2}$ learns information about the protected messages input by $\mathcal{S}_{S_2}$.

For part (3), the guarantees of garbled circuits (in particular, the fact that the labels cannot be distinguished from random strings) ensure that the probability of $\mathcal{A}_{S_2}$ decoding the output label is negligible in κ. And for part (4), note that as with the simulation against $\mathcal{A}_{S_1}$, the inputs are crafted to ensure that the output will match the decision output by $\mathcal{F}_{\text{reg-sh}}$. Thus, the real and ideal execution views are again indistinguishable to $\mathcal{A}_{S_2}$ except with probability negligible in κ.

If $\mathcal{S}_{S_2}$ receives accept in the registration phase, we proceed to simulation of the authentication phase, as follows:

(1) $\mathcal{S}_{S_2}$ samples random $\widehat{B}_2$ and sends it to $\mathcal{A}_{S_2}$.
(2) $\mathcal{S}_{S_2}$ constructs a garbled circuit as per Protocol 2 and sends to $\mathcal{A}_{S_2}$ the garbled gates $\mathcal{G}_f$.
(3) $\mathcal{A}_{S_2}$ and $\mathcal{S}_{S_2}$ engage $2m$ in OTB, with $\mathcal{S}_{S_2}$ entering bits $(0^m, 0^m)$. A s a result, $\mathcal{A}_{S_2}$ receives labels corresponding to $(B_2, \widehat{B}_2)$.
(4) $\mathcal{S}_{S_2}$ receives the output labels corresponding to (b_1, b_2) from $\mathcal{A}_{S_2}$.
(5) $\mathcal{S}_{S_2}$ sends to $\mathcal{A}_{S_2}$ the terminate signal.

Note that S_2's view in Protocol 2 is formed by its local randomness and the following components:

(1) S_2's share B_2 received in Protocol 1, which is used as an auxiliary input to this protocol.
(2) Receiving $\widehat{B}_2$ from C in Protocol 2 Step 1.
(3) Engaging in OTB with S_1 in Protocol 2 Step 3, with S_2 receiving $2m$ labels.
(4) Obtaining $\ell_{n-1}^{b_1}$ and $\ell_n^{b_2}$ from garbled circuit evaluation in Protocol 2 Step 3.
(5) Receiving the termination signal at the conclusion of Protocol 2.

The value in part (1) is output from registration and was discussed above. For part (2), $\mathcal{S}_{S_2}$ sends a random $\widehat{B}_2$ to $\mathcal{A}_{S_2}$, identically distributed to the value used in the protocol. For part (3), the security guarantees of OTB are such that there is a negligible in κ chance that $\mathcal{A}_{S_2}$ learns information about the protected messages input by $\mathcal{S}_{S_2}$ and thus cannot determine that $\mathcal{S}_{S_2}$ did not enter values according to the protocol specification (recall that $\mathcal{A}_{S_2}$ receives no authentication decision). And for part (4), the guarantees of garbled circuits again ensure that the probability of $\mathcal{A}_{S_2}$ decoding the output label is negligible in κ. Thus, the probability that the simulated view diverges from real execution is negligible in κ.

We also note that in all of the above cases, repeated authentication on the same biometric is still secure. That is, no substantial advantage can be gained by running a polynomial number of authentications, as the probability of leakage is negligibly small in the security parameter. This implies the inability of any server to succeed in reconstructing a biometric. □

Proof of Theorem 2. In the context of this theorem, security with respect to the servers is identical to that in the context of Theorem 1. Thus, we do not repeat security analysis in the presence of adversarial S_1 or S_2 and provide simulation to argue security in the presence of a malicious C, where C_{imp} initiates the authentication phase.

Adversary $\mathcal{A}_{C_{\text{imp}}}$. Recall that C_{imp} does not participate in the registration process for the given user and thus $\mathcal{A}_{C_{\text{imp}}}$ has no knowledge of the biometric B and the corresponding shares B_1 and B_2 entered by C_{auth} into the computation at registration time.

We build simulator $\mathcal{S}_{C_{\text{imp}}}$ for Protocol 2 that operates as follows:

(1) $\mathcal{S}_{C_{\text{imp}}}$ receives $\widehat{B}_1$ from $\mathcal{A}_{C_{\text{imp}}}$. If $\widehat{B}_1$ is not an m-bit string, $\mathcal{S}_{C_{\text{imp}}}$ signals abort.
(2) $\mathcal{S}_{C_{\text{imp}}}$ receives $\widehat{B}_2$ from $\mathcal{A}_{C_{\text{imp}}}$. If $\widehat{B}_2$ is not an m-bit string, $\mathcal{S}_{C_{\text{imp}}}$ signals abort.
(3) $\mathcal{S}_{C_{\text{imp}}}$ invokes $\mathcal{F}_{\text{auth-sh}}$ on input $\widehat{B}_1 \oplus \widehat{B}_2$ and receives accept or reject.
(4) $\mathcal{S}_{C_{\text{imp}}}$ sends the decision to $\mathcal{A}_{C_{\text{imp}}}$.

Note that client-server interaction in real and simulated executions is simple, with the client sending shares of its input and receiving authentication outcome. We next compare the adversarial views in real and simulated executions.

If the client does not form the shares $\widehat{B}_1$ and $\widehat{B}_2$ properly and instead sends values which are not m-bit strings, both real and simulated executions terminate with 100% probability. Otherwise, the shares are well-formed and will undergo identical verification to determine the outcome (either by querying the ideal functionality or performing the checks via garbled circuit evaluation). In both cases, the outcome will be reject if the input $\widehat{B}_1 \oplus \widehat{B}_2$ has not be properly normalized and/or is not within the required distance from the enrollment biometric B. Because the cryptographic tools used in the real execution are computationally secure, we obtain that there is only a negligible in κ probability that $\mathcal{A}_{C_{\text{imp}}}$ can distinguish the two views. □

Proof of Theorem 3. In the context of this theorem, we need to prove security in the presence of adversaries $\mathcal{A}_{S_1}$ and the combined $\mathcal{A}_{C_{\text{auth}} \cup S_2}$. Note that similarly to the proof of Theorem 1, C_{auth} does not gain any information by behaving maliciously in either the registration or authentication phases. And since a malicious C_{auth} has the ability to dictate the outcome of both phases, collusion with a malicious S_2 is trivial, as there is nothing they can do or learn that cannot be already achieved or provided by C_{auth}. For this reason, we only provide a simulation in the presence of $\mathcal{A}_{S_1}$. This will change for Theorem 4, where showing security in the presence of $\mathcal{A}_{C_{\text{imp}} \cup S_2}$ is needed.

Adversary $\mathcal{A}_{S_1}$. Our simulator $\mathcal{S}_{s_1}$ for Protocol 3, Reg-MAL, proceeds as follows:

(1) $\mathcal{S}_{S_1}$ invokes $\mathcal{F}_{\text{reg-mal}}$ and receives bits (b_1, b_2) and (B_1, c).
(2) If $b_1 = 1$, $\mathcal{S}_{S_1}$ chooses B_2 subject to $||B_1 \oplus B_2|| = 1$; otherwise, it chooses B_2 subject to $||B_1 \oplus B_2|| \neq 1$.
(3) $\mathcal{S}_{S_1}$ samples a κ_1-bit $\widetilde{v}$ at random from the witness space of the commitment scheme.
(4) If $b_2 = 1$, $\mathcal{S}_{S_1}$ computes $\widetilde{c} = \text{com}(B_1 \oplus B_2, \widetilde{v})$; otherwise, $\mathcal{S}_{S_1}$ sets $\widetilde{c} = c$.
(5) If $(b_1, b_2) = (1, 1)$, $\mathcal{S}_{S_1}$ stores (B_1, B_2) and $(\widetilde{c}, \widetilde{v})$ for use with authentication.
(6) $\mathcal{S}_{S_1}$ sends $(B_1, \widetilde{c})$ to $\mathcal{A}_{S_1}$.
(7) $\mathcal{S}_{S_1}$ receives from $\mathcal{A}_{S_1}$ the garbled gates $\mathcal{G}_f$.
(8) $\mathcal{S}_{S_1}$ and $\mathcal{A}_{S_1}$ engage in m instances of maliciously secure OTB and κ_1 instances of conventional maliciously secure OT, where $\mathcal{S}_{S_1}$ enters $(B_2, \widetilde{v})$ for its input. As a result, $\mathcal{S}_{S_1}$ receives labels corresponding to $(B_1 \oplus B_2, \widetilde{v})$.
(9) $\mathcal{S}_{S_1}$ also receives from $\mathcal{A}_{S_1}$ κ_2 labels corresponding to $\mathcal{A}_{S_1}$'s input $\widetilde{c}$.
(10) $\mathcal{S}_{S_1}$ evaluates the garbled circuit and sends the two output labels to $\mathcal{A}_{S_1}$.
(11) $\mathcal{S}_{S_1}$ receives the decision from $\mathcal{A}_{S_1}$ on behalf of C and S_2.

In order to compare the real and simulated views, we recall that S_1's view in Protocol 3 is formed by its local randomness and the following interactive components:

(1) Receiving B_1 and c from C in Protocol 3 Step 2.
(2) Engaging in OTB with S_2 in Protocol 3 Step 4, with S_2 receiving m labels.
(3) Engaging in OT with S_2 in Protocol 3 Step 4, with S_2 receiving κ_1 labels.
(4) Receiving $\ell_n^{b_1}$ and $\ell_{n-1}^{b_2}$ from S_2 in Protocol 3 Step 5.
(5) Using the labels to determine the computation outcome.

We next argue indistinguishability between real and simulated execution. For part (1) above, we have that S_1 receives B_1 and c from C in the protocol, while our simulator might supply S_1 with a modified $\widetilde{c}$, which differs from c supplied as input to the ideal functionality. Nevertheless, we argue that the adversary is unable to distinguish the values. In particular, when $b_2 = 0$ (i.e., the commitment c is not well formed or was not specified in a consistent way), the simulated view in this step is identical to the protocol view. In particular, client specified commitment c follows the same distribution as during protocol execution. However, when $b_2 = 1$ (i.e., the commitment is well formed and verifies), the simulator forms a different commitment $\widetilde{c}$ and communicates $(B_1, \widetilde{c})$ to S_1. In that case, the commitment is drawn from the same distribution as during the protocol execution and the views cannot be distinguished.

For parts (2) and (3), the simulator relies on security of oblivious transfer and a computationally limited adversary is unable to distinguish the simulation from the real protocol interaction.

Once S_1 receives the output labels of garbled circuit evaluation, it is able to interpret them and extract the output (b_1, b_2); this is parts (4) and (5) above. We argue that the inputs to garbled circuit evaluation were constructed by simulator $\mathcal{S}_{S_1}$ in such a way that the output bits will correspond to (b_1, b_2) provided to $\mathcal{S}_{S_1}$ by ideal functionality $\mathcal{F}_{\text{reg-mal}}$, which correspond to true output. In particular, if $\mathcal{S}_{S_1}$ receives $b_1 = 1$ (i.e., the input biometric is properly normalized), it uses (B_1, B_2) subject to $||B_1 \oplus B_2|| = 1$; otherwise B_2 is chosen to have $||B_1 \oplus B_2|| \neq 1$. This means that S_1 will recover the correct bit. Similarly, if $\mathcal{S}_{S_1}$ receives $b_2 = 1$ (the commitment verifies), it constructs another valid commitment $\widetilde{c}$ for which it knows the opening $(B_1 \oplus B_2, \widetilde{v})$. If $\mathcal{S}_{S_1}$ receives $b_2 = 0$ (invalid commitment), it uses the original commitment c with a random opening $(B_1 \oplus B_2, \widetilde{v})$, which will fail the verification with overwhelming probability, as desired. This means that during the simulated view, S_1 will recover the same (b_1, b_2) as during the protocol execution with all but negligible probability in the security parameter. This means the real and simulated views are indistinguishable.

If $\mathcal{S}_{S_1}$ receives $(b_1, b_2) = (1, 1)$ during registration, we proceed with simulation of Protocol 4, Auth-MAL, as follows:

(1) $\mathcal{S}_{S_1}$ invokes $\mathcal{F}_{\text{auth-mal}}$ and receives output bits (b_1, b_2, b_3).
(2) $\mathcal{S}_{S_1}$ samples random $\widehat{B}_1$ and sends $\widehat{B}_1$ to $\mathcal{A}_{S_1}$.
(3) Using B_1 and B_2 from registration simulation, $\mathcal{S}_{S_1}$ chooses $\widehat{B}_2$ as follows:
   (a) If $b_1 = 1$, $\widehat{B}_2$ is chosen subject to $\text{dist}(B_1 \oplus B_2, \widehat{B}_1 \oplus \widehat{B}_2) < t$; otherwise, it is chosen subject to $\text{dist}(B_1 \oplus B_2, \widehat{B}_1 \oplus \widehat{B}_2) \geq t$.
   (b) If $b_2 = 1$, $\widehat{B}_2$ is chosen subject to $||\widehat{B}_1 \oplus \widehat{B}_2|| = 1$; otherwise, it is chosen subject to $||\widehat{B}_1 \oplus \widehat{B}_2|| \neq 1$.
(4) In addition, $\mathcal{S}_{S_1}$ uses stored $\widetilde{v}$ to set v' as follows: if $b_3 = 1$, $v' = \widetilde{v}$; otherwise, $v' \neq \widetilde{v}$.
(5) $\mathcal{S}_{S_1}$ receives from $\mathcal{A}_{S_1}$ the garbled gates $\mathcal{G}_f$ and κ_2 labels corresponding to $\mathcal{A}_{S_1}$'s input c.
(6) $\mathcal{A}_{S_1}$ and $\mathcal{S}_{S_1}$ engage in $2m$ instances of maliciously secure OTB and κ_1 instances of conventional maliciously secure OT, with $\mathcal{S}_{S_1}$ entering $B_2, \widehat{B}_2, v'$. As a result, $\mathcal{S}_{S_1}$ receives labels corresponding to $(B_1 \oplus B_2, \widehat{B}_1 \oplus \widehat{B}_2, v')$.
(7) $\mathcal{S}_{S_1}$ evaluates the garbled circuit, obtains the output labels corresponding to (b_1, b_2, b_3), and sends them to $\mathcal{A}_{S_1}$.
(8) $\mathcal{S}_{S_1}$ receives the final decision from $\mathcal{A}_{S_1}$ on behalf of C and S_2.

This time, S_1's view in Protocol 4 is formed by its local randomness and the following components:

(1) S_1's share B_1 and commitment value c received in Protocol 3, which is used as an auxiliary input to this protocol.
(2) Receiving $\widehat{B}_1$ from C in Protocol 4 Step 1.
(3) Engaging in OTB and OT with S_2 in Protocol 4 Step 3, with S_2 receiving $2m + \kappa_1$ labels.
(4) Receiving labels $\ell_{n-1}^{b_1}$, $\ell_n^{b_2}$, and $\ell_n^{b_3}$ from S_2 in Protocol 4 Step 4.
(5) Using the labels to determine the computation outcome.

Parts (1)–(3) are similar to our prior analysis, where either there is no difference in the views or the views could be distinguished only with a negligible probability. To demonstrate indistinguishability for parts (4)–(5), we need to show that S_1 recovers the same output bits (b_1, b_2, b_3) as what the ideal functionality $\mathcal{F}_{\text{auth-mal}}$ (and real execution) supply.

The bits b_1 and b_2 correspond to the distance and normalization checks, respectively. The biometric shares ($\widehat{B}_1$ and $\widehat{B}_2$) are set to produce the correct bits during garbled circuit evaluation. However, bit b_3 is new and corresponds to the result of the commitment

check. To this extent, the simulator $\mathcal{S}_{S_1}$ uses consistent commitment opening information when $b_3 = 1$ and modifies $\widetilde{v}$ otherwise to cause the check to fail (which will happen with overwhelming probability). We obtain that S_1 will recover the correct output and is unable to distinguish the real and simulated views with more than a negligible probability. □

Proof of Theorem 4. In the context of this proof, showing security in the presence of semi-honest $\mathcal{A}_{S_1}$ follows the same procedures as in the proof of Theorem 3. Although $\mathcal{F}_{\text{auth-mal}}$ obtains inputs from C_{imp} instead of C_{auth}, the difference does not impact simulator construction and therefore the conclusions of Theorem 3 apply here as well.

We thus proceed to show security in the presence of malicious $\mathcal{A}_{C\cup S_2}$. There are two adversarial structures to consider: S_2 colluding with C_{auth} and S_2 colluding with C_{imp}. The latter is the most important to consider and show that a malicious user working in coordination with S_2 is unable to attack the original user's biometric B. The situation when the adversary is $\mathcal{A}_{S_2\cup C_{\text{auth}}}$, however, is different. During registration, this constitutes the identical scenario of that of Theorem 3, where the adversary is unable to learn any additional information. Then during authentication, $\mathcal{A}_{C_{\text{auth}}\cup S_2}$'s capabilities are that of S_2 alone who does not learn any output from the protocol. Therefore, we concentrate on showing security in the presence of adversary $\mathcal{A}_{C_{\text{imp}}\cup S_2}$.

Adversary $\mathcal{A}_{C_{\text{imp}}\cup S_2}$. We begin by constructing simulator $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ for Protocol 3, the registration. Because C_{imp} is not an active participant of the protocol, $\mathcal{A}_{C_{\text{imp}}\cup S_2}$'s capabilities during registration are equivalent to that of $\mathcal{A}_{S_2}$.

(1) $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ invokes $\mathcal{F}_{\text{reg-mal}}$ and receives (B_2, v) and accept or reject decision.
(2) $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ chooses $\widetilde{B}_1 \leftarrow \{0,1\}^m$ subject to $||\widetilde{B}_1 \oplus B_2|| = 1$ and computes $\widetilde{c} = \text{com}(\widetilde{B}_1 \oplus B_2, v)$.
(3) $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ sends (B_2, v) to $\mathcal{A}_{C_{\text{imp}}\cup S_2}$.
(4) $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ produces garbled circuit $\mathcal{G}_f$ as in Protocol 3 and sends it to $\mathcal{A}_{C_{\text{imp}}\cup S_2}$.
(5) $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ and $\mathcal{A}_{C_{\text{imp}}\cup S_2}$ engage in m instances of maliciously secure OTB and κ_1 instances of conventional maliciously secure OT, with $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ entering $\widetilde{B}_1$ into OTB. As a result, $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ receives labels corresponding to $(\widetilde{B}_1 \oplus B_2, v)$. $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ also sends labels $\{\ell_{m+\kappa_1+i}^{\widetilde{c}[i]}\}_{i\in[1,\kappa_2]}$ (that corresponds to $\widetilde{c}$) to $\mathcal{A}_{C_{\text{imp}}\cup S_2}$.
(6) $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ receives labels $\ell_{n-1}^{b_2}$ and $\ell_n^{b_1}$ from $\mathcal{A}_{C_{\text{imp}}\cup S_2}$.
(7) If $\ell_{n-1}^{b_2} = \ell_{n-1}^1$ and $\ell_n^{b_1} = \ell_{n-1}^1$, $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ stores (B_2, v) for use in authentication and sends the accept or reject decision received from to $\mathcal{A}_{C_{\text{imp}}\cup S_2}$. Otherwise, it sends reject to $\mathcal{A}_{C_{\text{imp}}\cup S_2}$.

S_2's view in Protocol 3 is formed by S_2's local randomness and the following components:

(1) Receiving (B_2, c) from C in Protocol 3 Step 2.
(2) Receiving garbled gates $\mathcal{G}_f$ from S_1 in Protocol 3 Step 3.
(3) Engaging in OTB with S_1 in Protocol 3 Step 4, with S_2 receiving m labels.
(4) Engaging in OT with S_1 in Protocol 3 Step 4, with S_2 receiving κ_1 labels.
(5) Receiving κ_2 labels $\ell_{m+\kappa_1+i}^{c[i]}$ from S_1 in Protocol 3 Step 4.
(6) Obtaining $\ell_n^{b_1}$ and $\ell_{n-1}^{b_2}$ from garbled circuit evaluation in Protocol 3 Step 5.
(7) Receiving the registration decision from S_1.

Communication in part (1) has identical distribution in real and simulated executions. That is, the value v is passed unmodified from $\mathcal{F}_{\text{reg-mal}}$ as it was obtained from C. The value B_2 is also generated according to the specification, as it would in the case of non-adversarial C.

Parts (2)–(5) in the simulation are executed as in real computation. Due to the security properties of garbled circuit evaluation and OT, $\mathcal{A}_{C_{\text{imp}}\cup S_2}$ is unable to obtain any information about $\widetilde{B}_1$ and $\widetilde{c}$ that the simulator enters into the circuit. Similarly, the labels in part (6) have identical distributions in real and simulated views and do not reveal any information to $\mathcal{A}_{C_{\text{imp}}\cup S_2}$.

Part (7) is the most interesting case. There are three situations to consider:

(1) If $\mathcal{A}_{C_{\text{imp}}\cup S_2}$ evaluates the circuit correctly, it will arrive at labels ℓ_n^1 and ℓ_{n-1}^1 because the inputs into the circuit were set to pass both checks. In that case, the simulator outputs the accept/reject decision (which is dictated by the client's inputs) and is identical to that in real execution.
(2) If $\mathcal{A}_{C_{\text{imp}}\cup S_2}$ instead modifies the values it enters into the garbled circuit (i.e., B_2, v, or both), the verification will not pass with overwhelming probability and the simulator outputs reject. This is identical to the behavior in the real execution.
(3) Lastly, if $\mathcal{A}_{C_{\text{imp}}\cup S_2}$ does not evaluate the circuit and returns malformed labels which do not correspond to the labels for circuit wires n and $n-1$, the simulator will output reject, which is the same as in the protocol execution.

We obtain that the real and simulated view differ with at most negligible probability and are therefore indistinguishable.

Provided registration was successful, we continue on to the authentication phase. To that end, we will need to extract input from this adversary. Note that for garbled circuits secure against (only) semi-honest adversaries, there is no provision for the evaluator to guarantee that each garbled gate computes the function prescribed to it. We can leverage this fact to allow $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ to extract input from $\mathcal{A}_{C_{\text{imp}}\cup S_2}$ as follows: for each input bit, there exists at least one path through $\mathcal{G}_f$ to one of the output bits. Hence for each bit $i \in [1, 2m+\kappa_1]$ of $\mathcal{A}_{C_{\text{imp}}\cup S_2}$'s input, we have $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ construct a circuit $\tilde{\mathcal{G}_f}^i$ where each gate along the guaranteed path is set to be an identity gate. From this $\mathcal{S}_{C_{\text{imp}}\cup S_2}$ can learn a single bit of $\mathcal{S}_{C_{\text{imp}}\cup S_2}$'s input, and then rewind and repeat. Since $\mathcal{A}_{C_{\text{imp}}\cup S_2}$ learns not the bit, but a pseudo-random label for which recovery is intractable, it cannot distinguish this execution from one which actually computes the prescribed function f.

We note that in what follows the notation v' represents the value the adversary inputs into the garbled circuit in the authentication phase, whereas v is the value output from registration, and these need not be the same given an active adversary. The simulation of Protocol 4 proceeds as follows:

(1) $\mathcal{S}_{C_{\text{imp}} \cup S_2}$ receives $\widehat{B}_1$ from $\mathcal{A}_{C_{\text{imp}} \cup S_2}$.
(2) $\mathcal{S}_{C_{\text{imp}} \cup S_2}$ uses the input extraction technique described above to obtain $(B_2, \widehat{B}_2, v')$.
(3) $\mathcal{S}_{C_{\text{imp}} \cup S_2}$ constructs garbled circuit $\mathcal{G}_f$ as per Protocol 4 and sends it to $\mathcal{A}_{C_{\text{imp}} \cup S_2}$.
(4) $\mathcal{S}_{C_{\text{imp}} \cup S_2}$ engages in malicious-secure OTB and OT with $\mathcal{A}_{C_{\text{imp}} \cup S_2}$, entering $B_1 = \widehat{B}_1 = 0$. As a result, $\mathcal{A}_{C_{\text{imp}} \cup S_2}$ receives labels corresponding to $(B_2, \widehat{B}_2, v')$.
(5) $\mathcal{S}_{C_{\text{imp}} \cup S_2}$ sends labels $\{\ell^0_{2m+\kappa_1+i}\}_{i \in [1,\kappa_2]}$ corresponding to $c = 0$ to $\mathcal{A}_{C_{\text{imp}} \cup S_2}$.
(6) $\mathcal{S}_{C_{\text{imp}} \cup S_2}$ sends S_2's inputs (B_2, v') and C_{imp}'s input $\widehat{B}_1 \oplus \widehat{B}_2$ to $\mathcal{F}_{\text{auth-mal}}$ and receives (accept, terminate), (reject, terminate), or (abort, abort) outputs for C and S_2, respectively.
(7) $\mathcal{S}_{C_{\text{imp}} \cup S_2}$ receives $(\ell^{b_3}_{n-2}, \ell^{b_2}_{n-1}, \ell^{b_1}_{n})$ from $\mathcal{A}_{C_{\text{imp}} \cup S_2}$.
(8) If each of the received labels corresponds to a valid label for the associated wire, $\mathcal{S}_{C_{\text{imp}} \cup S_2}$ sends to $\mathcal{A}_{C_{\text{imp}} \cup S_2}$ the decision it received from $\mathcal{F}_{\text{auth-mal}}$. Otherwise, it sends (abort, abort) to $\mathcal{A}_{C_{\text{imp}} \cup S_2}$.

S_2 and C's view in Protocol 4 is formed by S_2's and C's local randomness and the following interaction with S_1:

(1) Receiving garbled gates $\mathcal{G}_f$ from S_1 in Protocol 4 Step 2.
(2) Engaging in OTB with S_1 in Protocol 4 Step 3, with S_2 receiving $2m$ labels.
(3) Engaging in OT with S_1 in Protocol 4 Step 3, with S_2 receiving κ_1 labels.
(4) Receiving κ_2 labels $\ell^{c[i]}_{m+\kappa_1+i}$ from S_1 in Protocol 4 Step 3.
(5) Obtaining labels $\ell^{b_1}_{n}$, $\ell^{b_2}_{n-1}$, and $\ell^{b_3}_{n-2}$ from garbled circuit evaluation in Protocol 4 Step 4.
(6) Receiving the authentication decision from S_1.

This time, parts (1)–(5) correspond to garbled circuit and OT execution and, as before, do not disclose any information about inputs on which the circuit is evaluated. This means that $\mathcal{A}_{C_{\text{imp}} \cup S_2}$ is unable to determine the fact that the simulator enters zero values into the circuit.

For part (6), we note that the simulator submits to the ideal functionality $\mathcal{F}_{\text{auth-mal}}$ inputs extracted from the adversary as well as information consistent with the information produced by $\mathcal{F}_{\text{reg-mal}}$ at registration time. This means that, if the circuit is evaluated correctly, its output will correspond to the result of evaluation of same function as in real protocol execution. We see that in that case the simulator simply passes the output from the ideal functionality $\mathcal{F}_{\text{auth-mal}}$ to the adversary.

Otherwise, $\mathcal{A}_{C_{\text{imp}} \cup S_2}$ might deviate from circuit evaluation and return one of more labels which do not correspond to the output wires. In that case, the simulator sends to the corrupt parties abort, which is the identical behavior to that in the protocol execution.

We obtain that the adversary is unable to tell the real and simulated views apart with more than a negligible probability. □