Game-Set-MATCH: Using Mobile Devices for Seamless External-Facing Biometric Matching

Shashank Agrawal
Western Digital
shashank.agrawal@wdc.com

Saikrishna Badrinarayanan
Visa Research
sabadrin@visa.com

Pratyay Mukherjee
Visa Research
pratmukh@visa.com

Peter Rindal
Visa Research
perindal@visa.com

ABSTRACT

We use biometrics like fingerprints and facial images to identify ourselves to our mobile devices and log on to applications everyday. Such authentication is *internal-facing*: we provide measurement on the same device where the template is stored. If our personal devices could participate in *external-facing* authentication too, where biometric measurement is captured by a nearby external sensor, then we could also enjoy a frictionless authentication experience in a variety of physical spaces like grocery stores, convention centers, ATMs, etc. The open setting of a physical space brings forth important privacy concerns though.

We design a suite of secure protocols for external-facing authentication based on the cosine similarity metric which provide privacy for *both* user templates stored on their devices *and* the biometric measurement captured by external sensors in this open setting. The protocols provide different levels of security, ranging from passive security with some leakage to active security with no leakage at all. With the help of new packing techniques and zero-knowledge proofs for Paillier encryption—and careful protocol design, our protocols achieve very practical performance numbers. For templates of length 256 with elements of size 16 bits each, our fastest protocol takes merely 0.024 seconds to compute a match, but even the slowest one takes no more than 0.12 seconds. The communication overhead of our protocols is very small too. The passive and actively secure protocols (with some leakage) need to exchange just 16.5KB and 27.8KB of data, respectively. The first message is designed to be *reusable* and, if sent in advance, would cut the overhead down to just 0.5KB and 0.8KB, respectively.

1 INTRODUCTION

Biometric authentication has become a part of our daily lives. We use biometrics like fingerprints and facial images to identify ourselves to our mobile devices and log into mobile applications everyday [appb, appa, and, sam]. When we set up a device (enrollment phase) that supports biometric authentication, we provide several biometric measurements so that it can generate a high quality template. After that, we authenticate by providing a fresh biometric measurement. If the measurement matches with the stored template, the authentication succeeds (matching phase). We could think of such authentication as *internal-facing*: biometric data is collected for matching on the *same* device where the template is stored.

External-facing. The ability of mobile devices to store and process biometric data securely could be utilized for *external-facing* authentication too, where a sensor placed *outside* of the device (i.e. external sensor) collects biometric data for matching. Consider a large service provider with several physical locations in a region. A user would enroll her template with the provider through its app on her mobile device. Then, sensors placed at the provider locations could capture her biometric data and match it against the template stored on her device.[1] If the match is successful, then she could get a certain level of access, allowed to accomplish a certain task, etc., based on her account with the provider.

Places like amusement parks, convention centers, theaters, etc., could use this type of biometric matching at the entrance to quickly and conveniently identify people who have bought tickets earlier through their app. Grocery stores could use it to identify the right payment credentials and enable a smooth checkout. ATM machines could use it to identify the account to withdraw money from. Apartments could use it to only allow the right people in.

A very important benefit of this approach is that the biometric templates of the user population are *not* collected and stored in some giant database, whose breach would be disastrous [bio19, cbp19]. Instead, the templates stay on the devices of the users to which they belong.

Seamless experience. In the settings discussed above, there is immense value in making the user experience seamless. Ideally, users should not need to reach out for their devices and interact with them every time the service provider tries to authenticate them at its locations. Once a user demonstrates her trust in a provider by *explicitly* connecting with it and participating in the matching process at a location, the user device should *automatically* connect with the provider any time a match is needed at that location (and perhaps at other locations too). Google's Nearby Connections API [goo], for instance, could be used for this purpose. It uses both Bluetooth and WiFi Direct to connect devices.

Privacy concerns. Though a seamless experience seems quite useful and appealing, it constrains how a secure matching protocol can be designed. Let us take the example of an amusement park where people have lined up at the entrance. A sensor installed here would automatically connect to several user devices in proximity (if these devices are enrolled). If the sensor takes a biometric measurement of Alice (e.g., by capturing her picture), it *cannot* just share the

[1] The captured biometric data would be used to derive another template, which will be matched against the template stored on the device. However, to avoid confusion, we will use template to only refer to the device template. We will avoid an explicit mention of (the derivation of) the template on the sensor-side.

measurement with all the connected devices because it does not know which of them belongs to Alice.

Then, should the devices share their templates with the sensor? This is undesirable for at least two reasons. First, a template may be a lot more representative of an individual than a measurement collected by the sensor. Second, there may be people in the vicinity of the sensor who do not wish to enter the park but their devices are connected. These devices have no reason to just send the template to the sensor.

To summarize, neither the sensor should share the measurement collected with the devices in plaintext nor the devices should share the templates with the sensor in plaintext. Thus, we need to design a matching protocol between sensor and devices that can work with encrypted data and leaks as little information as possible about the sensitive biometric information involved.

Active attacks. While preventing information leakage is certainly important, attackers could also try to subvert the matching protocol. An attacker Bob could steal Alice's device, which stores her template, and try to trick the sensor into believing that it captured Alice's biometric, when the biometric is actually Bob's. In other words, Bob attempts to force the matching protocol to succeed when it should not, which would enable him to impersonate Alice. At an ATM machine, for example, he would be able to withdraw Alice's money.

Efficiency. Another crucial design aspect is that of efficiency, best exemplified by places like amusement parks where hundreds of people wait in line to enter. Any secure matching protocol for such use-cases should be both computation and communication efficient. The protocol should return the match outcome as quickly as possible. Perhaps it could start *before* a person gets to the front of the line, i.e., before a biometric measurement is taken. Furthermore, the protocol should involve a small amount of data exchange because the connection could be low-bandwidth like Bluetooth or data usage could be expensive. On the other hand, enrollment of biometric templates could be on the slower side because it will happen infrequently.

There exists an extensive literature on secure protocols for biometric matching [KAMR04, ST07, EFG+09, SSW09, HMEK11, BG11, OPJM10] for various modalities (fingerprint, iris, face, etc.) and distance measures (Hamming distance, Euclidean distance, etc.). However, to the best of our knowledge, all prior work is in the semi-honest model and does not consider active attacks. They are often motivated by the identification setting, where a measurement must be compared with a database of templates, as opposed to the authentication setting that we consider here (see Appendix A.1). As a result, they try to batch process as many comparisons as possible instead of optimizing a single comparison operation. Also, some of the papers lack a formal security model and/or a formal proof of the claims. A clear specification of the leakage, if any, could be missing too.

1.1 Our contribution

In this paper, we build efficient protocols for secure external-facing authentication from mobile devices with the help of new packing techniques and zero-knowledge proofs. We implement the protocols and study their performance in terms of running time and bandwidth usage. In more detail, our contributions can be described as follows:

Formalization. We formally specify the security goals of this type of biometric authentication with the help of a simulation-based definition in the ideal-real paradigm. We consider three entities in the security model: a device $\mathcal{D}$, a terminal $\mathcal{T}$ (which has an attached sensor) and a service provider $\mathcal{SP}$. We consider two phases in authentication: an enrollment phase where a user's biometric template is registered with a service provider, and a matching phase where a device and a terminal interact with each other to compare biometric data.

In the setting of external-facing authentication where a seamless experience is quite important, facial recognition seems to be the ideal choice at the moment. Lately, the cosine similarity metric has been quite popular in this domain (e.g. CosFace [WWZ+18], SphereFace [LWY+17], FaceNet [SKP15]) and could be used elsewhere too. So we represent biometric templates as vectors and our protocols compute the inner-product between them (in a secure manner). We assume that devices and terminals have the proper setup to capture facial images, process them to extract interesting features, and derive a template in a vector form. We do not consider this setup any further in this paper.

We treat the service provider and the terminal as semi-honest (passive) entities, but account for the possibility that they may *collude* with each other to learn more information about users' biometric templates. In other words, we provide a simulator for the *joint* view of a semi-honest $\mathcal{SP}$ and $\mathcal{T}$. On the other hand, we consider *both* semi-honest and malicious (active) devices.

Service providers could be malicious too but attempting to change the outcome of matching is not very useful for them (like it is for devices). If a paying member of an organization is refused entry to one of its locations, for instance, due to a biometric match failure, she could just ask for a refund. If a grocery store attempts to make the matching succeed with someone's device who is not checking out, the person could dispute the charge on her credit card.

Certainly, an active attack could also be used to gather more information about user data than any passive attack would allow. However, we do not consider active corruption of service providers and terminal in the paper, leaving it for future work. (Active corruption of mobile devices, as pointed out before, is indeed considered.)

Packing. We use Paillier encryption scheme [Pai99] to encrypt templates and use its homomorphic properties to operate on them. Typically, a single Paillier ciphertext can contain 2048 or 3072 bits of plaintext data but for biometric tasks, like the one we have here, the size of a plaintext value is very small. We introduce a new optimization technique for Paillier encryption that allows to batch encryptions of several plaintexts (of relatively small size) into one single ciphertext thereby reducing the computation and communication overhead of the basic encryption scheme considerably. (Some related works [EFG+09, BG11] rely on Damgard et al.'s [DGK07] encryption scheme for faster encryption and smaller ciphertexts than Paillier. However, decryption requires computing a discrete log.)

We note that our packing technique is significantly different from prior art [HMEK11, OPJM10] due to the support for multiplying two packed values. To get some insight, suppose we have two vectors $\overrightarrow{u} = (u_1, \ldots, u_\ell)$ and $\overrightarrow{v} = (v_1, \ldots, v_\ell)$, and we pack n out of ℓ elements into one ciphertext somehow. If we only care about summing up these vectors component-wise, we just need to make sure that there is some extra room or padding between the packed elements to capture the overflow. Amount of padding certainly depends on the number of additions we wish to perform.

However, a packing technique that supports component-wise multiplication requires more care. Multiplying two ciphertexts packed in a simple manner would produce terms where the desired products (of the form $u_i v_i$) are mixed with cross-products (of the form $u_i v_j$, $i \neq j$); so the packing scheme needs to be more *sparse*. Furthermore, the cross-products could leak extra information when packed ciphertexts are decrypted, so they need to be masked properly.

We design a new encoding scheme that takes care of these problems (Section 4), and lends us 8x savings in the semi-honest protocol and 6x savings in the malicious ones.

Zero-knowledge proofs. Zero-knowledge proofs are often used to provide security against active attacks. In our protocols, we would be worried about malformed ciphertexts, invalid templates, etc. We present several efficient zero-knowledge protocols for this (Section 8). Our first protocol is a zero-knowledge proof-of-knowledge of the plaintext values in a set of n Paillier ciphertexts. One can use Cramer et al. [CDN01]'s Σ−protocol to prove knowledge of each plaintext value individually, but this would make the communication complexity linear in n. We provide a new batched version of their protocol where the prover sends the same number of messages as the basic protocol. After applying the Fiat-Shamir heuristic, we get a protocol with a constant communication complexity (independent of n).

We also a need a proof for the well-formedness of templates, represented as vectors. We need to show in zero-knowledge that a vector has a certain L2 norm. Existing protocols [Lin17] for this have an overhead of κn for a statistical security parameter κ, say 40. We present new techniques to reduce the overhead to (roughly) n, thus achieving close to 40x speedup.

Protocols. We exploit the new packing and proof techniques to build a suite of efficient protocols for biometric matching based on the cosine similarity metric with trade-offs between security and efficiency (Section 5, 6, 7):

- Protocol I provides security against semi-honest attacks, and leaks inner product to the device.
- Protocol II provides security against malicious devices and semi-honest service-provider/terminal, and leaks inner product to the terminal.
- Protocol III provides security in the same setting as the above but leaks no information whatsoever.

Each of the protocols consists of an enrollment phase and a matching phase. In the enrollment phase, a provider signs an *encrypted* template provided by a device. The device hides the template but convinces the provider that it is well-formed using the zero-knowledge proofs discussed above.

The signed encrypted template is the first message sent by the device in a matching phase with a terminal. The signature convinces the terminal that the template is well-formed without having to engage in another proof session. Since an encrypted template leaks no information whatsoever to the terminal, the first message of the matching protocol could be sent even before the terminal had a chance to record a measurement.

Protocol I, II and III have three, four and five rounds of communication, respectively, in the matching phase. A round consists of a message sent from the device to the terminal or vice versa. If the first message is sent early on, then the matching phase in all the three protocols would have one fewer round.

Apart from zero-knowledge proofs, we use cryptographic tools like message authentication codes, oblivious transfer and garbled circuits to achieve security under active attacks in the second and third protocols. The first protocol only relies on the security of Paillier encryption.

Implementation & evaluation. We implement the three protocols above and evaluate them on a single commodity laptop. We use 3072-bit Paillier, estimated to have 112 bits of computational security, and a statistical security parameter of 40 bits. We consider various template lengths $\ell \in \{128, 256, 512, 1024\}$ combined with different bit lengths $m \in \{8, 16, 24\}$ for each template element. We discuss here some performance numbers for $\ell = 256$ and $m = 16$. Please refer to Section 9 for full details.

For Protocol I, enrollment takes 0.6 seconds while the matching phase requires just 0.024 seconds. This can be explained by the fact that Paillier encryption requires a large exponentiation while decryption and homomorphic operations are relatively efficient. The first message of matching phase (encrypted template) is only 16KB as opposed to 128KB, thanks to packing. Thus, packing helps us achieve an improvement of 8x here. Rest of the protocol only requires the terminal to communicate 516 bytes. As a result, Protocol I has very low communication overhead.

Despite considering malicious devices, Protocol II and III require only 5 seconds for enrollment, due to our efficient zero-knowledge proofs. The matching phase of Protocol II, which is the one that will be run again and again, takes just 0.05 seconds. The first message here is of size 27KB as opposed to 162KB, a saving of 6x due to packing. Rest of the protocol consumes only 0.8KB.

The most secure protocol, Protocol III, is also quite fast. It takes only 0.12 seconds to compute a match. The communication overhead is on the higher side though: a total of 142KB is exchanged as opposed to 28KB for Protocol II, due to the use of garbled circuits. With some assistance from the service provider in a preprocessing phase, this can be brought down to as little as 20KB. See Section 9 for details.

One can also use LWE-based additive homomorphic encryption schemes instead of Pailler encryption in our protocols. We discuss this alternative in some detail in Appendix A.3.

1.2 Related Work

Numerous works have considered the problem of biometric authentication. Some are based on the idea of deriving a cryptographic key from noisy biometric measurements [DRS04, Boy04, KSvdV+05, CRC+19]. One shortcoming of this approach is the difficulty in

deriving a consistent key despite potential errors in the biometric measurement. Another line of work uses techniques from secure multi-party computation to evaluate the traditional matching algorithms on encrypted data [KAMR04, ST07, EFG+09, SSW09, HMEK11, BG11, OPJM10]. Our work lies in this second category.

To the best of our knowledge, all prior work is in the semi-honest model and does not consider active attacks. Most of them are motivated by the identification setting where a measurement must be compared with a database of templates [EFG+09, SSW09, HMEK11, BG11, OPJM10, ASD+16], and only a few earlier ones directly consider an authentication-like setting [KAMR04, ST07]. Due to this, the emphasis is generally on optimizations that help improve the efficiency of several simultaneous comparisons, as opposed to a single one.

We could have several simultaneous authentication attempts from different users in our setting too, but the number of such attempts would generally be small. Moreover, we cannot afford to increase the latency of a single comparison for a better overall throughput because in our use-cases, users expect to learn outcomes as soon as possible.

We refer to Appendix A for a discussion of authentication versus identification as well as a detailed look at the papers above.

2 PRELIMINARIES

Notation. We denote the security parameter by λ. We represent biometric values using vectors: the device's biometric template is represented as a vector $\overrightarrow{u} = (u_1, \ldots, u_\ell)$ and the terminal's biometric template is represented as a vector $\overrightarrow{w} = (w_1, \ldots, w_\ell)$. Here, for all $i \in [\ell]$, $u_i, w_i \in \mathbb{Z}_{2^m}$ (i.e., u_i, w_i are m-bit numbers). We use $[\![x]\!]$ to denote an encryption of x. Sometimes, we use $[\![x; r]\!]$ to denote that message x is encrypted using randomness r.

Definition 1. ***(Cosine Similarity)*** *For vectors $\overrightarrow{u}, \overrightarrow{w}$, the Cosine Similarity between them is a value between 0 and 1 defined as follows:*

$$\mathsf{CS.Dist}(\overrightarrow{u}, \overrightarrow{w}) = \frac{\langle \overrightarrow{u}, \overrightarrow{w} \rangle}{||\overrightarrow{u}|| \cdot ||\overrightarrow{w}||}.$$

Note that if the L2-norm of both vectors $||\overrightarrow{u}||, ||\overrightarrow{w}||$ is ζ, $\mathsf{CS.Dist}(\overrightarrow{u}, \overrightarrow{w}) = \langle \overrightarrow{u}, \overrightarrow{w} \rangle / \zeta^2$. To check if $\mathsf{CS.Dist}(\overrightarrow{u}, \overrightarrow{w}) \geq T'$ for some $T' \in [0, 1]$, it is sufficient to check that $\langle \overrightarrow{u}, \overrightarrow{w} \rangle \geq T$ where $T = T' \cdot \zeta^2$.

We formally define zero knowledge proofs, digital signatures, oblivious transfer and garbled circuits in Appendix B.

2.1 Paillier Encryption

The additively homomorphic encryption scheme by Paillier [Pai99] (Plr.Setup, Plr.Enc, Plr.Add, Plr.Dec) is as follows:

- $\mathsf{Plr.Setup}(\lambda) =: (\mathsf{epk}, \mathsf{esk})$: Choose random primes p and q of length λ. Set $N = pq, g = (N+1), d = \mathsf{lcm}(p-1, q-1)$. Compute $\mu = L(g^d \bmod N^2)^{-1} \bmod N$ where $L(\cdot)$ is defined as $L(x) = \frac{x-1}{N}$. Output public key $\mathsf{epk} = (N, g)$ and secret key $\mathsf{esk} = (d, \mu)$.
- $\mathsf{Plr.Enc}(\lambda, x, \mathsf{epk}) =: [\![x; r]\!]$: Given plaintext $x \in \mathbb{Z}_N$, output ciphertext $[\![x; r]\!] = g^m r^N \bmod N^2$ where r is chosen randomly in $\mathbb{Z}_N^*$.
- $\mathsf{Plr.Add}(\lambda, [\![x]\!], [\![y]\!]) =: [\![x + y]\!]$: Given ciphertexts $[\![x]\!]$ and $[\![y]\!]$, output ciphertext $[\![x + y]\!] = [\![x]\!] \cdot [\![y]\!] \bmod N^2$.
- $\mathsf{Plr.Dec}(\lambda, c, \mathsf{esk}) =: x$: Given ciphertext c and secret key esk, output $x = L(c^d \bmod N^2) \cdot \mu \bmod N$.

We describe the correctness and security properties satisfied by this scheme in Appendix B.

3 BIOMETRIC AUTHENTICATION

In this section, we formally define secure biometric authentication. We use the ideal-real world paradigm to capture the security goals.

Ideal functionality. The ideal functionality $\mathcal{F}$ for biometric authentication is shown in Figure 1. It has two query types. The first allows a user to register a biometric template $\overrightarrow{u}$ via her device $\mathcal{D}$ with a service provide $\mathcal{SP}$ in an 'enrollment' phase. Subsequently, the device can engage in a 'matching' session with a terminal $\mathcal{T}$ using her registered template. The terminal provides a measurement $\overrightarrow{w}$. If $\overrightarrow{u}$ and $\overrightarrow{w}$ are close enough, then $\mathcal{F}$ reports that there was a match.

In addition, the functionality explicitly allows a pair of leakage functions to be defined. We require these because some of our protocols reveal more information than just the predicate of whether there was a match or not. Looking forward, some of our protocols will reveal the inner product between $\overrightarrow{w}$ and $\overrightarrow{u}$ to one of the parties.

Parameters: The functionality $\mathcal{F}$ is parameterized by values $\ell, m \in \mathbb{N}$, $\zeta \in \mathbb{Z}_{2^{2m}\ell}$, a distance function $\mathsf{CS.Dist} : \mathbb{Z}_{2^m}^\ell \times \mathbb{Z}_{2^m}^\ell \to [0, 1]$, a threshold $T \in [0, \zeta^2]$, and leakage functions $L_{\mathsf{dev}}, L_{\mathsf{term}}$ both mapping $\mathbb{Z}_m^\ell \times \mathbb{Z}_m^\ell$ to $\mathbb{Z}$. It interacts with three parties $\mathcal{D}$, $\mathcal{SP}$ and $\mathcal{T}$ via the following queries.
Queries:

- **On receiving a query of the form (**"Enrollment", sid, $\overrightarrow{u}$, $\mathcal{SP}$**) from** $\mathcal{D}$: Only if sid is unmarked, check if $\|\overrightarrow{u}\| = \zeta$. If the check succeeds, send ("Enrolled", sid, $\mathcal{D}$) to $\mathcal{SP}$ and $\mathcal{D}$, store (sid, $\overrightarrow{u}$, $\mathcal{D}$, $\mathcal{SP}$), and mark sid as "Enrolled". Otherwise, send ("Failed", sid, $\mathcal{D}$) to $\mathcal{SP}$ and $\mathcal{D}$, and mark sid as "Failed".
- **On receiving a query of the form (**"Match", sid, ssid, $\mathcal{T}$**) from** $\mathcal{D}$: Check whether sid is marked and there exists a record of the form (sid, $\overrightarrow{u}$, $\mathcal{D}$, $\mathcal{SP}$). If the check fails, send ("Failed", sid, ssid, $\mathcal{D}$) to $\mathcal{T}$ and $\mathcal{D}$. Else:
  (1) send (sid, ssid, $\mathcal{D}$) to $\mathcal{T}$;
  (2) receive (sid, ssid, $\overrightarrow{w}$, $\mathcal{D}$) from $\mathcal{T}$ where $\|\overrightarrow{w}\| = \zeta$;
  (3) if $\langle \overrightarrow{u}, \overrightarrow{w} \rangle \geq T$, (i.e., $\mathsf{CS.Dist}(\overrightarrow{u}, \overrightarrow{w}) \geq T/\zeta^2$) send (sid, ssid, 1) to $\mathcal{D}$ and $\mathcal{T}$, else send (sid, ssid, 0) to them. Also send $L_{\mathsf{dev}}(\overrightarrow{u}, \overrightarrow{w})$ to $\mathcal{D}$ and $L_{\mathsf{term}}(\overrightarrow{u}, \overrightarrow{w})$ to $\mathcal{T}$.

Figure 1: Ideal Functionality $\mathcal{F}$

Real world. A protocol π for biometric authentication consists of two phases: an enrollment phase and a matching phase. The enrollment phase is between a device $\mathcal{D}$ and a service provider $\mathcal{SP}$. Once $\mathcal{D}$ is enrolled, it can participate in a matching session with a terminal $\mathcal{T}$. Typically, the enrollment phase will be run once for a device, while the matching phase will be run many times, possibly with different terminals.

Threat model. We consider semi-honest (honest-but-curious) service providers and terminals in this paper, but account for the real possibility that they may *collude* with each other to learn more information about the biometric templates of users (beyond the fact

that there was a match or not, and any leakage). In other words, we provide a simulator for the *joint* view of a semi-honest $\mathcal{SP}$ and $\mathcal{T}$.

On the other hand, we consider both semi-honest and malicious devices. A semi-honest device would try to learn more information about the measurements than it is supposed to. A malicious device would go further: it could try to enroll an invalid template ($\|\overrightarrow{u}\| \neq 1$), make the matching succeed even when the measurement is not close ($\langle \overrightarrow{u}, \overrightarrow{w} \rangle < T$), etc.

Security. We say that a protocol π is secure if for every adversary $\mathcal{A}$ in the real world (of the type described above), there exists a simulator $\mathcal{S}$ in the ideal world such that the joint distribution of the output of honest parties and view of $\mathcal{A}$ in the real world is computationally indistinguishable from the joint distribution of the output of honest parties and $\mathcal{S}$ in the ideal world.

4 PACKING

Recall that Paillier is an additively homomorphic encryption scheme with a plaintext space $\mathbb{Z}_{pq}$ where p, q are large primes. Typical sizes of p, q is 1024 bits each which means that a single Paillier ciphertext can contain 2048 bits of plaintext data. The security level would then be similar to RSA with a 2048 bit key. However, in several applications, the size of a plaintext value is very small. In our case, for instance, it is only 8 or 16 bits. We cannot make the ciphertext smaller due to security constraints, so encrypting each plaintext value separately results in a huge communication overhead.

In this section, we introduce a packing technique for Paillier encryption that allows one to batch encryptions of several plaintexts (of relatively small size) into one single ciphertext, thereby reducing the communication overhead of the basic encryption scheme. We design new encoding, decoding, addition and multiplication algorithms to effectively achieve this task. Each of these algorithms are applied to *plaintext* data.

Notation. Let m denote the size in bits of each plaintext value we wish to encrypt. Let n be the number of plaintext values we want to encode together into one ciphertext. Let L denote the size in bits of each ciphertext (which is 2048 in the case of Paillier). Let d denote the length in bits of the padding applied to each plaintext value. We elaborate more on d later in the section. We also consider two levels for the encodings. At level-1, we have plaintext encodings where no multiplication operation has been performed. We can multiply two level-1 encodings to produce a level-2 encoding. We also refer to a level-2 encoding as a hyper-encoding. Two level-2 encodings *cannot* be multiplied together but can be added. This emulates the linear homomorphism of Paillier encryption.

4.1 Algorithms, Properties & Use

We now describe the algorithms for packing and their properties.

Definition 2 (Packing Scheme). *An (m, n, L, d) packing scheme consists of the following algorithms. (λ is always an implicit input.)*

- *Encoding & Decoding. There are two ways to encode/decode denoted by* $(\mathsf{Encode}_1, \mathsf{Decode}_1)$ *and* $(\mathsf{Encode}_2, \mathsf{Decode}_2)$. *Encode algorithms,* Encode_1 *and* Encode_2*, take m-bit strings $u_1, \ldots, u_n$ as input.* Encode_1 *outputs a level-1 encoding u^*, while* Encode_2 *outputs a level-2 encoding $\widetilde{u}$ (hyper-encoded value). The encodings are L-bit strings. Decode algorithms,* Decode_1 *and* Decode_2*, take an L-bit string as input and output n number of m-bit strings.*
- *Addition. There are two addition algorithms, one for each type of encoding.* Add_1 *adds two level-1 encoded values u^*, v^* to produce another (level-1) value w^*.* Add_2 *is similar but operates on level-2 encodings.*
- *Multiplication. There is only one multiplication algorithm* Mult. *It takes two* level-1 *encodings u^*, v^* and outputs a* level-2 *encoding $\widetilde{w}$.*
- *Masking. There is only one randomized algorithm* Mask. *It transforms one level-2 encoding into another.*

Correctness. Informally, correctness requires that decoding an encoded value at either level must produce the same output as performing the corresponding (component-wise) addition and multiplication operations on the underlying vector of plaintext values. For the mask procedure, we require the following: for any $\widetilde{u}$, $\mathsf{Mask}(\widetilde{u})$, $\mathsf{Decode}_2(\widetilde{u}) = \mathsf{Decode}_2(\mathsf{Mask}(\widetilde{u}))$.

Security. Informally, we require that the distribution produced by the output of the masking algorithm is statistically indistinguishable from one produced by a simulator that is given only the decoding of this output. Formally, for all λ, there exist a PPT algorithm Sim such that for all hyper-encoding $\widetilde{w}$, the following two ensembles are statistically indistinguishable:

$$\{\mathsf{Mask}(\widetilde{w})\} \text{ and } \left\{\mathsf{Sim}\Big(\mathsf{Decode}_2(\mathsf{Mask}(\widetilde{w}))\Big)\right\}.$$

Using packing in Paillier. We will use the packing scheme to "pack" a number of plaintexts together into one ciphertext. In particular, we will pack n plaintexts $u_1, \ldots, u_n$ together. We will use Encode_1 to generate a level-1 encoding u^* first. Then we encrypt u^* to produce a Paillier ciphertext $[\![u^*]\!]$. If addition is the only operation performed on the ciphertext, the underlying encoding remains at level-1. Say, after one addition of two ciphertexts $[\![u^*]\!]$ and $[\![v^*]\!]$, we obtain $[\![w^*]\!]$. From correctness, we have that $\mathsf{Decode}_1(w^*) =: \overrightarrow{w}$ where $w_i = u_i + v_i$ for all $i \in [n]$.

For multiplication with a known value, the product will produce a level-2 encoding. In particular, to multiply $[\![u^*]\!]$ by plaintext $\overrightarrow{v}$, one needs to first encode $\overrightarrow{v}$ into v^* (i.e. $v^* := \mathsf{Encode}_1(\overrightarrow{v})$) and then use Paillier scalar multiplication to produce $[\![\widetilde{w}]\!]$ such that $\mathsf{Decode}_2(\widetilde{w}) = u_1 v_1 + \ldots u_n v_n$. Furthermore, we need to use the masking algorithm after each multiplication for security reasons to produce $[\![\widetilde{z}]\!]$ where $\widetilde{z} \leftarrow \mathsf{Mask}(\widetilde{w})$, and so $\mathsf{Decode}_2(\widetilde{z}) = \mathsf{Decode}_2(\widetilde{w})$. As guaranteed by the security property, a masked value does not have any additional information beyond its decoding, so it is now "safe" to decrypt $[\![\widetilde{z}]\!]$ before decoding.

Linear procedures. We note that certain operations are performed on the ciphertexts homomorphically. Since Paillier encryption supports only additive homomorphism we need to guarantee that those procedures are linear, in that only addition of ciphertexts and multiplying a ciphertext with a plaintext are required to homomorphically perform the operations. Looking ahead, our construction ensures that all procedures Add_1, Add_2, Mult, Mask[2] are linear and can be supported over Paillier ciphertexts.

[2]The masking algorithm requires sampling random coins. However, this does not need the ciphertext. We only need to add the random coins to the encoding that is encrypted which can be done using additive homomorphism.

Algorithms:

- $\mathsf{Encode}_1(u_1, \ldots, u_n)$: Given $u_1, \ldots, u_n \in \mathbb{Z}_{2^m}$, output $u^* = \sum_{i=1}^{n} 2^{k\cdot(2^{i-1}-1)} u_i \in \mathbb{Z}_{2^{k^2 n}}$.
- $\mathsf{Add}_1(u^*, v^*)$: Output $w^* = u^* + v^*$.
- $\mathsf{Decode}_1(u^*)$: Write u^* as $u^* = \sum_{i=1}^{n} 2^{k\cdot(2^{i-1}-1)} u_i$, where $u_i \in \mathbb{Z}_{2^k}$, and output $(u_1, \ldots, u_n)$.
- $\mathsf{Mult}(u^*, v^*)$: Output $\widetilde{w} = u^* \cdot v^*$.
- $\mathsf{Encode}_2(u_1, \ldots, u_n)$: Given $u_1, \ldots, u_n \in \mathbb{Z}_{2^m}$, output $\widetilde{u} = \sum_{i=1}^{n} 2^{k\cdot 2\cdot(2^{i-1}-1)} u_i \in \mathbb{Z}_{2^{k^2 n}}$.
- $\mathsf{Add}_2(\widetilde{u}, \widetilde{v})$: Output $\widetilde{w} = \widetilde{u} + \widetilde{v}$.
- $\mathsf{Mask}(\widetilde{w})$: Output $\widetilde{z} = \left(\widetilde{w} + \sum_{j\in\bar{I}} 2^{j\cdot k}\rho_j\right)$ where $I = \{0, 2, 6, (2^n - 2)\}$ and the ρ_j's are picked randomly in $\mathbb{Z}_{2^k}$.
- $\mathsf{Decode}_2(\widetilde{u})$: Write $\widetilde{u}$ as $\left(\sum_{i\in I} 2^{i\cdot k} u_i\right) + \left(\sum_{j\in\bar{I}} 2^{j\cdot k}\rho_j\right)$ where $u_i, \rho_j \in \mathbb{Z}_{2^k}$ and $I = \{0, 2, \ldots, (2^n - 2)\}$. Output $\{u_i\}_{i\in I}$.

Here, $k = (m + d + \lambda)$ and $L = (k \cdot 2^n)$.

Correctness. Correctness follows easily by inspection.

Security. Given a tuple $(w_1, \ldots, w_n)$ computed as $\mathsf{Decode}_2(\mathsf{Mask}(\widetilde{w}))$, the simulator outputs a hyper-encoding $\widetilde{z}$ as follows:

- Let $I = \{0, 2, \ldots, (2^n - 2)\}$. Set $\{z_i\}_{i\in I} = (w_1, \ldots, w_n)$.
- $\widetilde{z} = \left(\sum_{i\in I} 2^{i\cdot k} z_i\right) + \left(\sum_{j\in\bar{I}} 2^{j\cdot k}\rho_j\right)$ where each ρ_j is picked uniformly at random.

It is easy to observe that $\{\mathsf{Mask}(\widetilde{w})\}$ and $\{\mathsf{Sim}(w_1, \ldots, w_n)\}$ are identically distributed and hence statistically indistinguishable.

Figure 2: Packing Scheme

4.2 Construction

Our construction is described in Figure 2. Below, we give a detailed overview of our techniques.

Level-1 Encoding. Consider n plaintexts $u_1, \ldots, u_n \in \mathbb{Z}_{2^m}$ that we wish to encode together and later encrypt into one ciphertext. The naive way to encode would be to create a plaintext $u^* = \sum_{i=1}^{n} 2^{(i-1)(m+d)} u_i \in \mathbb{Z}_{2^{n(m+d)}}$ where there are d empty padding bits between every consecutive pair of values that are encoded. Let k denote the size of each block in the encoding, i.e., $k = (m + d)$.

Level-1 Addition. It is easy to see that given two such encoded values u^*, v^*, adding them directly gives the encoded version of their sum. That is,

$$u^* + v^* = \sum_{i=1}^{n} 2^{(i-1)(m+d)}(u_i + v_i) \in \mathbb{Z}_{2^{n(m+d)}}$$

as long as d is large enough to prevent overflows. The d padding bits are present to ensure that if $(u_i + v_i) \geq 2^m$, this overflow will still be stored in the same block (within the padding bits) and not spill into the block used for storing $(u_{i+1} + v_{i+1})$. In the case of just one addition, $d = 1$ will suffice. Similarly, multiplying u^* by a single value $x \in \mathbb{Z}_{2^m}$ also works directly. That is,

$$xu^* = \sum_{i=1}^{n} 2^{(i-1)(m+d)}(xu_i).$$

Once again, we need to ensure that d is large enough so that $x \cdot u_i \leq (2^k - 1)$. In this case, $d = m$ suffices. More generally, d must be large enough to prevent overflows when performing a combination of these operations. We will elaborate on the choice of d later.

Multiplication. Multiplying two level-1 encodings component wise does not work directly as in the above operations. Consider a scenario where you generate an encoding u^* of $\overrightarrow{u}$ and then wish to multiply its components with the corresponding components of an encoding v^* of $\overrightarrow{v}$. Looking ahead, this scenario arises when one party encrypts u^* and sends it to another party who wishes to perform multiplications homomorphically between the vectors $\overrightarrow{u}$ and $\overrightarrow{v}$. For example, if we have $n = 2$ then

$$v^*u^* = 2^{0\cdot k} \cdot v_1u_1 + 2^{1\cdot k} \cdot (v_1u_2 + v_2u_1) + 2^{2\cdot k} v_2u_2.$$

So, we do not get $2^{0\cdot k}v_1u_1 + 2^{1\cdot k}v_2u_2$ as desired. However, observe that the values of interest (v_1u_1, v_2u_2) are stored in the result, just with v_2u_2 at a different block position than in the input encoding. It is easy to verify that for any n,

$$v^*u^* = v_1u_1 + 2^k(\ldots) + 2^{2\cdot k}(\ldots) + \ldots + 2^{(2n-2)\cdot k} v_nu_n.$$

v_1u_1 and v_nu_n can still be recovered from the first and last blocks. However, the other $(\ldots)$ blocks will not be *pure*, i.e., they will be mixed with products of the form u_iv_j for $i \neq j$.

We now show a new encoding strategy which allows us to recover all of the products v_iu_i from various positions of v^*u^*. For $n = 3$, we now encode u_1, u_2, u_3 as

$$u^* = u_1 + 2^k u_2 + 2^{3\cdot k} u_3$$

Note that the block at position 2 (the term $2^{2\cdot k}$) is set to zero. If we then multiply two encodings v^* and u^*, we obtain:

$$v_1u_1 + 2^k(v_1u_2 + v_2u_1) + 2^{2\cdot k} v_2u_2 + 2^{3\cdot k}(v_1u_3 + v_3u_1) + 2^{4\cdot k}(v_2u_3 + v_3u_2) + 2^{5\cdot k}(0) + 2^{6\cdot k} v_3u_3.$$

Observe that the desired products can now be recovered from block positions 0, 2 and 6. Generalizing this to any arbitrary n, the crucial idea is to inductively place each u_i, v_i at a block position such that the product term u_iv_i gets placed in a block that does not have any other products. This is achieved by encoding u_i at block position $(2^{i-1} - 1)$. To be more clear, we now encode $u_1, u_2, \ldots, u_n$ as $u^* = u_1 + 2^k u_2 + \ldots + 2^{k\cdot(2^{n-1}-1)} u_n$ That is,

$$\mathsf{Encode}_1(u_1, \ldots, u_n) = u^* = \sum_{i=1}^{n} 2^{k\cdot(2^{i-1}-1)} u_i \in \mathbb{Z}_{2^{k^2 n}}$$

If we multiply two such encodings, the blocks at positions 0, 2, 6, $\ldots$, $(2^n - 2)$ will have the desired component-wise product terms.

Here, k denotes the size of each block, i.e. $k = (m + d)$. We must pick a large enough d to ensure that none of the blocks overflow. Each block contains a sum of at most n terms each of which is a product of two m bit values. Therefore, $d = m + \log n + 1$ to perform one such component-wise multiplication.

Also, observe that u^* is technically contained within $\mathbb{Z}_{2^{k^2 n-1}}$. Instead, we refer to an encoding as contained in $\mathbb{Z}_{2^{k^2 n}}$ because when we multiply two level-1 encodings, the resulting product (a level-2 encoding) is contained in $\mathbb{Z}_{2^{k^2 n}}$ now. To maintain consistency in notation between the ranges of both encodings, we use the same term $\mathbb{Z}_{2^{k^2 n}}$ for both.

Masking. In the above multiplication computation, the resultant level-2 encoding $\widetilde{w} = \mathsf{Mult}(u^*, v^*)$ also has cross-product terms such as $u_1 v_2$, $v_2 u_1$, etc. (or a sum of such terms) in the clear which would be revealed during decoding, thereby leaking more information than just component-wise multiplication. We overcome this issue by the use of algorithm Mask which takes a hyper-encoding $\widetilde{w}$ as input and adds a uniformly random value of size $(m + d + \lambda)$ bits to each of the blocks that may have cross-product terms. Before we describe the exact details of how this is done, the first thing to note is that to account for this, when computing a level-1 encoding (which is later multiplied to produce this hyper-encoding $\widetilde{w}$), we now increase the size of each block by λ bits. That is, we now set k to be $m + d + \lambda$.

Let I denote the set of block positions in $\widetilde{w}$ which have the values we care about. From the previous multiplication scenario, these would correspond to the actual component wise products in $\widetilde{w}$. For example, $I = \{0, 2, 6, \ldots, (2^n - 2)\}$. (We note that more compact packings are possible but it is difficult to succinctly describe them.) The algorithm Mask adds a random value of length $k = m + d + \lambda$ to each block not in I. Concretely, given a hyper-encoding $\widetilde{w}$, $\mathsf{Mask}(\cdot)$ does the following: Compute

$$\widetilde{z} = \widetilde{w} + \sum_{j \in \overline{I}} 2^{j \cdot k} \rho_j$$

where ρ_j is a uniformly random value in $\mathbb{Z}_{2^k}$.

To ensure that adding a ρ_i term in a block does not corrupt the value in other blocks, we require that the addition does not overflow the $(m + d + \lambda)$ bits of the given block. Recall we guarantee that $(m + d)$ is large enough to hold the sum of the terms. Therefore, the probability of an overflow is at most the probability that the most significant λ bits of ρ_i are all ones, which is negligible.

Observe that in a sequence of operations that involve, for example, a multiplication of two level-1 encodings followed by additions with other hyper-encodings, it is sufficient to perform the masking only once over the final hyper-encoding before decoding it.

Level-2 Encoding & Addition. From the above overview, the level-2 encoding and addition follow naturally:

$$\widetilde{u} = \mathsf{Encode}_2(u_1, \ldots, u_n) = \sum_{i=1}^{n} 2^{k \cdot 2 \cdot (2^{i-1} - 1)} u_i \in \mathbb{Z}_{2^{k 2^n}},$$

$$\mathsf{Add}_2(\widetilde{u}, \widetilde{v}) = (\widetilde{u} + \widetilde{v}) = \sum_{i=1}^{n} 2^{k \cdot 2 \cdot (2^{i-1} - 1)} (\widetilde{u_i} + \widetilde{v_i}) \in \mathbb{Z}_{2^{k 2^n}}.$$

Choice of Parameters. Give m and L (the size of each value and that of overall encoding, respectively), we need to set the values of n and d (number of elements that can be packed together and the length of padding, respectively) by ensuring that $(k \cdot 2^n) \leq L$ where $k = (m + d + \lambda)$. The choice of d depends on the operations performed.

Remark. In the rest of the paper, for ease of exposition, we invoke algorithms $\mathsf{Add}_1, \mathsf{Add}_2, \mathsf{Mult}$ on encodings, using regular "+", "·" symbols. We will always refer to level-1 encodings with an asterisk in the superscript (as in u^*) and a hyper-encoding with a tilde (as in $\widetilde{u}$).

5 ALL PARTIES SEMI-HONEST

In this section, we build a secure biometric authentication protocol with a three round matching phase[3] based on the additively homomorphic encryption scheme by Paillier [Pai99], which is based on the decisional composite residuosity assumption. Our construction is secure against a semi-honest adversary that either corrupts a device or jointly corrupts the service provider and the terminal. At the end of the matching phase, the inner product between the device and terminal vectors is leaked to the device. Formally, in Appendix C we prove the following theorem:

Theorem 1. *Assuming the security of Paillier encryption, Π_1 in Figure 3 is a three round biometric authentication protocol that is secure against any semi-honest adversary with $L_{\mathsf{dev}}(\overrightarrow{u}, \overrightarrow{w}) = \langle \overrightarrow{u}, \overrightarrow{w} \rangle$ and $L_{\mathsf{term}}(\overrightarrow{u}, \overrightarrow{w}) = \perp$.*

5.1 Construction

Let (Plr.Setup, Plr.Enc, Plr.Add, Plr.Dec) denote the Paillier encryption scheme and (Encode_1, Encode_2, Add_1, Add_2, Mult, Mask, Decode_1, Decode_2) denote the (m, n, L, d) packing scheme described in Section 4. We describe the construction of our three round secure protocol in Figure 3.

Overview. In the enrollment phase, the device $\mathcal{D}$ encodes its input vector $\overrightarrow{u} = (u_1, \ldots, u_\ell)$ to produce $t = \lceil \frac{\ell}{n} \rceil$ encodings $(u_1^*, \ldots, u_t^*)$. It encrypts these encodings using the Paillier encryption scheme and sends the ciphertexts $\{[\![u_i^*]\!]\}_{i \in [t]}$ to the service provider $\mathcal{SP}$. Since we only consider a semi-honest adversary, $\mathcal{SP}$ is guaranteed that $[\![u^*]\!]$ indeed well formed and has L2-norm of ζ.

The matching phase begins with $\mathcal{D}$ sending the same ciphertexts $\{[\![u_i^*]\!]\}_{i \in [t]}$ to terminal $\mathcal{T}$. $\mathcal{T}$ encodes its input vector $\overrightarrow{w}$ to produce encodings $(w_1^*, \ldots, w_t^*)$. Then, for each $i \in [t]$, $\mathcal{T}$ homomorphically multiplies encodings (u_i^*, w_i^*) to produce a level-2 encoding of the product. It then homomorphically adds all these ciphertexts to produce an encryption $[\![\widetilde{z}]\!]$ of the level-2 encoding of the inner product $\mathsf{IP} = \langle \overrightarrow{u}, \overrightarrow{w} \rangle$. To ensure that the level-2 encoding $\widetilde{z}$ doesn't leak any additional information, we rely on the security of the masking algorithm: $\mathcal{T}$ homomorphically masks $\widetilde{z}$ and sends $[\![\mathsf{Mask}(\widetilde{z})]\!]$. Note that the masking procedure can be performed homomorphically: the underlying randomness is sampled externally and can then be added homomorphically. $\mathcal{D}$ can then decrypt this ciphertext and decode the resulting hyper-encoding to learn IP. It then checks if $\mathsf{IP} > T$ to learn the output which is shared with $\mathcal{T}$. Observe that this protocol leaks IP to $\mathcal{D}$.

There is a subtle issue in the above protocol. Recall that while the security of the masking scheme prevents the decoding from leaking information about cross-product terms, $\mathcal{D}$ still ends up learning each component-wise term $(u_i \cdot w_i)$ of IP rather than just the inner product on performing the decoding. To prevent this, $\mathcal{T}$ adds a secret sharing of 0, i.e., adds a random value ρ_i to each component in such a way that $\sum_{i \in [n]} \rho_i = 0$. Therefore, each term of the decoded still looks uniformly random while they sum up to IP. We defer a formal proof to Appendix C.

Parameter Selection. We now briefly discuss the choice of the parameters used for packing. Recall from Section 4 that we need

[3] From now, when we mention rounds, we refer to the number of rounds in just the matching phase.

Enrollment Phase: The device $\mathcal{D}$ has an input vector $\overrightarrow{u} = (u_1, \ldots, u_\ell)$ such that $\|\overrightarrow{u}\| = \zeta$. The enrollment phase proceeds as follows.

- Device $\mathcal{D}$ does the following:
  (1) Compute $u_1^*, \ldots, u_t^*$ as the encoding of $\overrightarrow{u}$. That is, $u_1^* = \mathsf{Encode}_1(u_1, \ldots, u_n), \ldots, u_t^* = \mathsf{Encode}_1(u_{\ell-n+1}, \ldots, u_\ell)$ where $t = \lceil \frac{\ell}{n} \rceil$.
  (2) Compute $(\mathsf{esk}, \mathsf{epk}) \leftarrow \mathsf{Plr.Setup}(1^\lambda)$. For all $i \in [t]$, compute $[\![u_i^*]\!] = \mathsf{Plr.Enc}(\mathsf{epk}, u_i^*)$.
  (3) Send $(\mathsf{epk}, \{[\![u_i^*]\!]\}_{i\in[t]})$ to $\mathcal{SP}$.
- Service provider $\mathcal{SP}$ responds back with a message "Enrolled".

Matching Phase: The terminal $\mathcal{T}$ has an input vector $\overrightarrow{w} = (w_1, \ldots, w_\ell)$ such that $\|\overrightarrow{w}\| = \zeta$. The matching phase proceeds as follows.

- **Round 1:** $(\mathcal{D} \rightarrow \mathcal{T})$ Device $\mathcal{D}$ sends $(\mathsf{epk}, \{[\![u_i^*]\!]\}_{i\in[t]})$ to $\mathcal{T}$.
- **Round 2:** $(\mathcal{T} \rightarrow \mathcal{D})$ $\mathcal{T}$ does the following:
  (1) Compute $(w_1^*, \ldots, w_t^*) = \mathsf{Encode}_1(\overrightarrow{w})$.
  (2) Compute $[\![\widetilde{z}]\!] = [\![\sum_{i=1}^t (u_i^* \cdot w_i^*)]\!]$ using the algorithm $\mathsf{Plr.Add}(\cdot)$. Note that $\widetilde{z}$ is now a level-2 encoding.
  (3) Pick n random k-bit strings $\rho_1, \ldots, \rho_n$ such that $(\rho_1 + \ldots + \rho_n) = 0$. Let $\widetilde{\rho} = \mathsf{Encode}_2(\rho_1, \ldots, \rho_n)$.
  (4) Compute and send $[\![\widetilde{\mathsf{IP}}]\!] = [\![\mathsf{Mask}(\widetilde{z} + \widetilde{\rho})]\!]$ to $\mathcal{D}$.
- **Round 3:** $(\mathcal{D} \rightarrow \mathcal{T})$ $\mathcal{D}$ does the following:
  (1) Compute $(\mathsf{IP}_1, \ldots, \mathsf{IP}_n) = \mathsf{Decode}_2(\mathsf{Plr.Dec}(\mathsf{esk}, [\![\widetilde{\mathsf{IP}}]\!]))$. Set $\mathsf{IP} = \sum_{i=1}^n \mathsf{IP}_i$.
  (2) If $\mathsf{IP} \geq T$, $\mathcal{D}$ sends 1 to $\mathcal{T}$. Else, sends 0.
- **Output Computation.** Both parties output the bit sent in the last round.

Figure 3: Protocol Π_1

to pick the number of values n to encode based on the constraint that $k \cdot 2^n \leq L$, where L is the length of one Paillier ciphertext, $k = (m + d + \lambda)$, and m is the length of each feature u_i, w_i that are initially encoded. In this protocol, using any encoding u_i^* or w_i^*, we perform one multiplication with another encoding followed by $(t-1)$ additions with such encodings. So, d should be $(m + \log n + \log t + 2)$. Therefore, we need to satisfy the constraint that $2^n \cdot (2m + \log n + \log\lceil \ell/n \rceil + \lambda + 2) \leq L$ to determine how many elements n to pack together. This depends on the specific values of (λ, m, L, ℓ) that we consider and we refer to Section 9 for more details on our choice and implementation results.

6 MALICIOUS DEVICE

In this section, we build a secure biometric authentication protocol with a four round matching phase. Our protocol is based on digital signatures, Paillier encryption [Pai99] and honest-verifier zero knowledge proofs. We need proofs for three specific languages related to the encryption scheme in the enrollment phase (i) Proving that the public key was correctly generated (ii) batch proving knowledge of plaintexts given a set of ciphertexts and (iii) proving knowledge of the L2-norm of a vector given a set of ciphertexts encrypting it. We describe these zero knowledge proofs in Section 8. They are based on the security of the Paillier encryption scheme. At the end of the matching phase, the inner product between the device and terminal vectors is leaked to the terminal. Formally, we prove the following theorem:

Theorem 2. *Assuming the security of Paillier encryption and digital signatures, Π_2 in Figure 4 is a four round biometric authentication protocol that is secure against any adversary that can maliciously corrupt the device or jointly corrupt the terminal and authority in a semi-honest manner with $L_{\mathsf{dev}}(\overrightarrow{u}, \overrightarrow{w}) = \perp$ and $L_{\mathsf{term}}(\overrightarrow{u}, \overrightarrow{w}) = \langle \overrightarrow{u}, \overrightarrow{w} \rangle$.*

Paillier encryption is based on the decisional composite residuosity (DCR) assumption while digital signatures can be based on any one way function (which is implied by DCR). We refer to Appendix D for the formal security proof of Theorem 2.

6.1 Construction

Let (Gen, Sign, Verify) be a signature scheme. Let $\mathsf{IP}_{\mathsf{Max}} = 2^{2m}\ell$ denote the maximum possible value of the inner product $\langle \overrightarrow{u}, \overrightarrow{w} \rangle$ where $\overrightarrow{u}, \overrightarrow{w}$ are the biometric input vectors of the device and terminal, respectively, with norm ζ.

Overview. Unlike the previous protocol, here, a corrupt $\mathcal{D}$ can behave maliciously. In the enrollment phase, $\mathcal{D}$ sends encryptions $([\![u_1]\!], ..., [\![u_\ell]\!])$ of its input vector $\overrightarrow{u}$ without encoding them. Using the zero knowledge protocols, $\mathcal{D}$ also proves that these ciphertexts were honestly generated and the L2-norm of $\overrightarrow{u}$ is ζ. If the proofs verify successfully, $\mathcal{SP}$ now converts these into encryptions $[\![u_1^*]\!], \ldots, [\![u_t^*]\!]$ of encodings and sends it to $\mathcal{D}$ along with a signature on them.

$\mathcal{D}$ initiates the matching phase by sending these ciphertexts $[\![u_1^*]\!], \ldots, [\![u_t^*]\!]$ along with the signature in the first round. $\mathcal{T}$ verifies the signature to ensure that the underlying vector $\overrightarrow{u}$ was indeed enrolled earlier. As before, $\mathcal{T}$ encodes its input vector $\overrightarrow{w}$ to produce encodings $(w_1^*, \ldots, w_t^*)$. Unlike the previous protocol, since we do not want to leak the inner product $\mathsf{IP} = \langle \overrightarrow{u}, \overrightarrow{w} \rangle$ to $\mathcal{D}$, $\mathcal{T}$ does not compute $[\![\widetilde{z}]\!]$ where $\widetilde{z}$ is a level-2 encoding of IP. Instead, $\widetilde{z}$ is now an encoding of $(a \cdot \mathsf{IP} + b)$ where a and b are random values. As before, $\mathcal{T}$ adds random ρ_i values before masking and sends $[\![\mathsf{Mask}(\widetilde{z} + \widetilde{\rho})]\!]$ in the second round. $\mathcal{D}$ can recover $(a \cdot \mathsf{IP} + b)$ which is sent back to $\mathcal{T}$ in the third round. This allows $\mathcal{T}$ to learn IP as it knows (a, b). It then checks if $\mathsf{IP} > T$ to learn the output which is shared with $\mathcal{D}$ in the last round. Observe that this protocol leaks IP to $\mathcal{D}$.

We now discuss how to prevent a malicious $\mathcal{D}$ from cheating in the third round. Suppose it sends $X' \neq (a\mathsf{IP} + b)$. We modify $\mathcal{T}$'s strategy as follows: let $\mathsf{IP}' = \frac{X'-b}{a}$. Output $\perp$ if IP' is larger than the maximum possible value $\mathsf{IP}_{\mathsf{Max}}$ of the inner product (for any two norm-ζ vectors $\overrightarrow{u}, \overrightarrow{w}$). Since (a, b) are uniformly random, with overwhelming probability, a cheating $\mathcal{D}$ will be unable to pick $X' \neq (a\mathsf{IP}+b)$ with both of the following conditions: (i) $\mathsf{IP}' < \mathsf{IP}_{\mathsf{Max}}$ (ii) (a) $\mathsf{IP} \leq T$ and $\mathsf{IP}' > T$ (OR) (ii) (b) $\mathsf{IP}' \leq T$ and $\mathsf{IP} > T$.

Enrollment Phase: The device $\mathcal{D}$ has an input vector $\overrightarrow{u} = (u_1, \ldots, u_\ell)$ such that $\|\overrightarrow{u}\| = \zeta$. The enrollment phase proceeds as follows.

- Device $\mathcal{D}$ computes (esk, epk) $\leftarrow \mathsf{Plr.Setup}(1^\lambda)$ and sends (epk, $[\![u_1; \rho_1]\!]$, …, $[\![u_\ell; \rho_\ell]\!]$) to $\mathcal{SP}$.
- Run ZK protocols Π_{epk}, $\Pi_{\text{ptxt-knwldg}}$ and $\Pi_{\text{zk-L2}}$ (with $y = \zeta$) from Section 8 with $\mathcal{D}$ as prover and $\mathcal{SP}$ as verifier. For the first proof, the statement is epk and witness is the underlying primes (p, q). For the next two proofs, statement is (epk, $[\![u_1; \rho_1]\!]$, …, $[\![u_\ell; \rho_\ell]\!]$) and witness is $\{u_i, \rho_i\}_{i=1}^{\ell}$.
- If either proof fails, $\mathcal{SP}$ sends the message "Failed" to $\mathcal{D}$.
- Else, $\mathcal{SP}$ does the following:
  (1) Generate (sk, vk) $\leftarrow \mathsf{Gen}(1^\lambda)$.
  (2) Compute $([\![u_1^*]\!], \ldots, [\![u_t^*]\!])$ using algorithm $\mathsf{Plr.Add}(\cdot)$ where $u_1^*, \ldots, u_t^*$ is the encoding of $\overrightarrow{u}$. That is, $u_1^* = \mathsf{Encode}_1(u_1, \ldots, u_n), \ldots, u_t^* = \mathsf{Encode}_1(u_{\ell-n+1}, \ldots, u_\ell)$ where $t = \lceil \frac{\ell}{n} \rceil$.
  (3) Send $\sigma = \mathsf{Sign}(\mathsf{sk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!]))$ to $\mathcal{D}$ and store vk at the terminal $\mathcal{T}$.

Matching Phase: The terminal $\mathcal{T}$ has an input vector $\overrightarrow{w} = (w_1, \ldots, w_\ell)$ such that $\|\overrightarrow{w}\| = \zeta$. The matching phase proceeds as follows.

- **Round 1:** $(\mathcal{D} \rightarrow \mathcal{T})$ Device $\mathcal{D}$ sends $(\sigma, \mathsf{epk}, \{[\![u_i^*]\!]\}_{i\in[t]})$ to $\mathcal{T}$.
- **Round 2:** $(\mathcal{T} \rightarrow \mathcal{D})$ $\mathcal{T}$ does the following:
  (1) Output $\perp$ if $\mathsf{Verify}(\mathsf{vk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!]), \sigma) \neq 1$.
  (2) Compute $(w_1^*, \ldots, w_t^*) = \mathsf{Encode}(\overrightarrow{w})$. Pick strings $a \in \mathbb{Z}_\lambda$, $b \in \mathbb{Z}_{(\lambda \cdot \log \mathsf{IP}_{\mathsf{Max}} + \lambda)}$ uniformly at random. Let $\widetilde{b} = \mathsf{Encode}_2(b)$.
  (3) Compute $[\![\widetilde{z}]\!] = [\![\sum_{i=1}^{t} (u_i^* \cdot (a \cdot w_i^*)) + \widetilde{b}]\!]$ using the algorithm $\mathsf{Plr.Add}(\cdot)$.
  (4) Pick random k-bit strings $\rho_1, \ldots, \rho_n$ such that $\sum_{i\in[n]} \rho_i = 0$. Let $\widetilde{\rho} = \mathsf{Encode}_2(\rho_1, \ldots, \rho_n)$.
  (5) Compute and send $[\![\widetilde{X}]\!] = [\![\mathsf{Mask}(\widetilde{z} + \widetilde{\rho})]\!]$ to $\mathcal{D}$.
- **Round 3:** $(\mathcal{D} \rightarrow \mathcal{T})$ $\mathcal{D}$ computes $(X_1, \ldots, X_n) = \mathsf{Decode}_2(\mathsf{Plr.Dec}(\mathsf{esk}, [\![\widetilde{X}]\!]))$, sets $X = \sum_{i=1}^{n} X_i$, and sends X to $\mathcal{T}$.
- **Round 4:** $(\mathcal{T} \rightarrow \mathcal{D})$ $\mathcal{T}$ does the following:
  (1) Set $\mathsf{IP} = \frac{X-b}{a}$. If $\mathsf{IP}_{\mathsf{Max}} < \mathsf{IP}$, output $\perp$ (indicates that the device cheated).
  (2) If $T \leq \mathsf{IP} \leq \mathsf{IP}_{\mathsf{Max}}$, send 1 to $\mathcal{D}$. Else, send 0.
- **Output Computation.** Both parties output the bit sent in the last round.

Figure 4: Protocol Π_2

Parameter Selection. Using any encoding u_i^* or w_i^*, we first perform one multiplication with another encoding, a multiplication with a constant a of length λ, followed by $(t-1)$ additions of such encodings and then adding it to a level-2 encoding of b. So, d should be $(m + \log n + \log t + \lambda + 3)$. Therefore, we need to satisfy the constraint that $2^n \cdot (2m + \log n + \log\lceil \ell/n \rceil + 2\lambda + 3) \leq L$ to determine how many elements n to pack together.

7 MALICIOUS DEVICE AND NO LEAKAGE

In this section, we build a secure biometric authentication protocol with a five round matching phase. Our protocol is based on digital signatures, garbled circuits, two-message oblivious transfer (OT), Paillier encryption [Pai99] and honest-verifier zero knowledge proofs. We need proofs for specific languages related to the encryption scheme as in Section 6. The protocol leaks no extra information to any party. Formally, we prove the following theorem:

Theorem 3. *Assuming the security of Paillier encryption, digital signatures, garbled circuits and two message oblivious transfer, Π_3 in Figure 5 is a five round biometric authentication protocol that is secure against any adversary that can maliciously corrupt the device or jointly corrupt the terminal and authority in a semi-honest manner with no leakage to any party.*

Paillier encryption and two message OT (either in CRS or random oracle model) can be instantiated based on the DCR assumption while digital signatures and garbled circuits can be based on any one way function (which is implied by DCR). We refer to Appendix E for the formal security proof.

7.1 Construction

Let (Garble, Eval) denote a garbling scheme for circuits and let ($\mathsf{OT.Round}_1$, $\mathsf{OT.Round}_2$, OT.Output) be a two-message oblivious transfer protocol. We describe our five round protocol in Figure 5.

Overview. We build on the previous protocol to additionally ensure that the inner product IP is not leaked to the terminal $\mathcal{T}$. Recall from the previous protocol that the value $X = (a \cdot \mathsf{IP} + b)$ was directly sent by $\mathcal{D}$ to $\mathcal{T}$ which allowed $\mathcal{T}$ to recover IP and compare it with the threshold. Here, instead, we perform this comparison inside a garbled circuit. One way to do so would be for $\mathcal{T}$ to generate a garbled circuit that $\mathcal{D}$ can evaluate using input X. The circuit would internally compute $\mathsf{IP} = (X - b)/a$. However, performing a division inside a garbled circuit is not very efficient. To build a more efficient protocol, we could do the following: in the second round, $\mathcal{T}$ sends encryptions of (hyper-encodings of) $X = \mathsf{IP}$ and $Y = (a \cdot \mathsf{IP} + b)$. $\mathcal{D}$ can recover and feed both as inputs to the garbled circuit so that it only performs one multiplication to check that $Y = (a \cdot X + b)$.

While the random choice of (a, b) once again ensures that a malicious $\mathcal{D}$ can't cheat by providing wrong inputs (X, Y) to the circuit, notice that we now leak $X = \mathsf{IP}$ to $\mathcal{D}$! To solve this problem, $\mathcal{T}$ encrypts IP via a one-time pad to generate $X = (\mathsf{IP} + \mathsf{pad})$, thereby hiding IP from $\mathcal{D}$. That is, $\mathcal{D}$ now receives encryptions of hyper-encodings of $X = (\mathsf{IP} + \mathsf{pad})$ and $Y = (a \cdot \mathsf{IP} + b)$, and labels corresponding to inputs (pad, a, b). The circuit first removes the one-time pad to recover IP and then checks that $Y = (a \cdot \mathsf{IP} + b)$ as before. We use oblivious transfer (OT) to enable $\mathcal{D}$ to obtain the labels to evaluate the garbled circuit. The enrollment phase is same

Enrollment Phase: Same as the enrollment phase in Figure 4 in Section 6.
Matching Phase: The terminal $\mathcal{T}$ has an input vector $\overrightarrow{w} = (w_1, \ldots, w_\ell)$ such that $\|\overrightarrow{w}\| = \zeta$. The matching phase proceeds as follows.

- **Round 1:** $(\mathcal{D} \to \mathcal{T})$ Device $\mathcal{D}$ sends $(\sigma, \mathsf{epk}, \{[\![u_i^*]\!]\}_{i\in[t]})$ to $\mathcal{T}$.
- **Round 2:** $(\mathcal{T} \to \mathcal{D})$ $\mathcal{T}$ does the following:
  (1) Output $\perp$ if $\mathsf{Verify}(\mathsf{vk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!]), \sigma) \neq 1$.
  (2) Compute $(w_1^*, \ldots, w_t^*) = \mathsf{Encode}(\overrightarrow{w})$. Pick random strings $a \in \mathbb{Z}_\lambda$, $b \in \mathbb{Z}_{(\lambda\cdot\log \mathsf{IP}_{\mathsf{Max}}+\lambda)}$, $\mathsf{pad} \in \mathbb{Z}_{\log \mathsf{IP}_{\mathsf{Max}}+\lambda}$. Let $\widetilde{\mathsf{pad}} = \mathsf{Encode}_2(\mathsf{pad})$ and $\widetilde{\mathsf{b}} = \mathsf{Encode}_2(b)$.
  (3) Compute $[\![\widetilde{z}_0]\!] = [\![\sum_{i=1}^t (u_i^* \cdot w_i^*) + \widetilde{\mathsf{pad}}]\!]$ and $[\![\widetilde{z}_1]\!] = [\![\sum_{i=1}^t (u_i^* \cdot (a \cdot w_i^*)) + \widetilde{\mathsf{b}}]\!]$ using algorithm $\mathsf{Plr.Add}(\cdot)$.
  (4) Pick $2n$ random k-bit strings $\rho_1, \ldots, \rho_{2n}$ such that $(\rho_1 + \ldots + \rho_n) = 0$, $(\rho_{n+1} + \ldots + \rho_{2n}) = 0$. Let $\widetilde{\rho}_0 = \mathsf{Encode}_2(\rho_1, \ldots, \rho_n)$ and $\widetilde{\rho}_1 = \mathsf{Encode}_2(\rho_{n+1}, \ldots, \rho_{2n})$.
  (5) Compute and send $([\![\widetilde{X}]\!], [\![\widetilde{Y}]\!])$ to $\mathcal{D}$ where $[\![\widetilde{X}]\!] = [\![\mathsf{Mask}(\widetilde{z}_0 + \widetilde{\rho}_0)]\!]$ and $[\![\widetilde{Y}]\!] = [\![\mathsf{Mask}(\widetilde{z}_1 + \widetilde{\rho}_1)]\!]$.
- **Round 3:** $(\mathcal{D} \to \mathcal{T})$ $\mathcal{D}$ does the following:
  (1) Compute $(X_1, \ldots, X_n) = \mathsf{Decode}_2(\mathsf{Plr.Dec}(\mathsf{esk}, [\![\widetilde{X}]\!]))$, $(Y_1, \ldots, Y_n) = \mathsf{Decode}_2(\mathsf{Plr.Dec}(\mathsf{esk}, [\![\widetilde{Y}]\!]))$.
  (2) Set $X = \sum_{i=1}^n X_i$ and $Y = \sum_{i=1}^n Y_i$.
  (3) Generate and send $\mathsf{ot}^{\mathsf{rec}} \leftarrow \mathsf{OT.Round}_1((X, Y); \rho^{\mathsf{ot}})$ to $\mathcal{T}$.
- **Round 4:** $(\mathcal{T} \to \mathcal{D})$ $\mathcal{T}$ does the following:
  (1) Compute $(\widetilde{C}, \mathsf{lab}) = \mathsf{Garble}(C)$ for circuit C described in Figure 6. Let $(\mathsf{lab}_{\mathsf{pad}}, \mathsf{lab}_a, \mathsf{lab}_b)$ denote the labels for inputs (pad, a, b).
  (2) Let $\{\mathsf{lab}^0_{X,Y}, \mathsf{lab}^1_{X,Y}\}$ denote the set of labels for inputs (X, Y). Generate $\mathsf{ot}^{\mathsf{sen}} = \mathsf{OT.Round}_2(\ (\mathsf{lab}^0_{X,Y}, \mathsf{lab}^1_{X,Y}), \mathsf{ot}^{\mathsf{rec}})$.
  (3) Send $(\widetilde{C}, \mathsf{lab}_{\mathsf{pad}}, \mathsf{lab}_a, \mathsf{lab}_b, \mathsf{ot}^{\mathsf{sen}})$ to $\mathcal{D}$.
- **Round 5:** $(\mathcal{D} \to \mathcal{T})$ $\mathcal{D}$ does the following:
  (1) compute $\mathsf{lab} = \mathsf{OT.Output}(\mathsf{ot}^{\mathsf{sen}}, \mathsf{ot}^{\mathsf{rec}}, \rho^{\mathsf{ot}})$.
  (2) Run $\mathsf{Eval}(\widetilde{C}, \mathsf{lab}, \mathsf{lab}_{\mathsf{pad}}, \mathsf{lab}_a, \mathsf{lab}_b)$ to learn output bit y and label $\mathsf{lab}_{\mathsf{y}}$. Send $\mathsf{lab}_{\mathsf{y}}$ to $\mathcal{T}$.
- **Output Computation.** $\mathcal{D}$ outputs the bit y. $\mathcal{T}$ decrypts $\mathsf{lab}_{\mathsf{y}}$ and outputs the value.

Figure 5: Protocol Π_3

Inputs: $(X, Y, \mathsf{pad}, a, b)$.
Computation:

- Compute $\mathsf{IP} = (X - \mathsf{pad})$. If $(a \cdot \mathsf{IP} + b) \neq Y$, output $\perp$.
- If $T \leq \mathsf{IP}$, output 1. Else, output 0.

Figure 6: Circuit C to be garbled.

as the previous protocol with the only addition being $\mathcal{SP}$ also runs the setup of the OT protocol.

Parameter Selection. The analysis of the choice of parameters used for packing is almost identical to that in Section 6.

8 ZERO KNOWLEDGE PROOFS

In this section, we describe the two honest-verifier zero knowledge proofs for the Paillier encryption scheme that are used in our protocols. Recall that a Paillier ciphertext $[\![x]\!]$ is given by $g^x r^N \bmod N^2$, where the public parameters are $N = pq$ and $g = N + 1$, and private randomness is $r \in \mathbb{Z}_N^*$. Sometimes, we use notation $[\![x; r]\!]$ to explicitly denote that randomness r is used to generate the ciphertext.

8.1 Public key well-formedness

Hazay et al. [HMRT12, HMR+19] construct an honest-verifier zero knowledge protocol to show that a public key epk of the Paillier scheme was correctly generated. We denote this protocol by Π_{epk} and refer to section 3.3 in the full version of [HMR+19] for the construction and proof.

8.2 Knowledge of Plaintexts

In this section, we present a honest-verifier zero knowledge protocol to prove knowledge of the plaintexts and randomness underlying n Paillier ciphertexts $[\![x_1]\!] \ldots, [\![x_n]\!]$. Cramer et al. [CDN01] designed a Σ−protocol for proving knowledge of both the message and randomness for a single ciphertext $[\![x; r]\!]$. Here we provide a new batched version of their protocol that allows us to simultaneously prove the knowledge of n message-randomness pairs $(x_1, r_1), \ldots, (x_n, r_n)$ given n ciphertexts $[\![x_1; r_1]\!], \ldots, [\![x_n; r_n]\!]$ while only requiring the same amount of communication from the prover.

We achieve this generalization by leveraging the linear structure of their protocol. At a very high level, in the first round of their protocol, the prover sends $[\![s; u]\!]$ for a random s. Given a random challenge e, the prover has to respond back with $w = (s + ex) \bmod N$ and $z = ug^t r^e$ where t satisfies some linear equation with respect to (s, w, e, x). The verifier then checks that the ciphertext $[\![w; z]\!]$ obtained by encrypting message w with randomness z is equal to the product of $[\![s; u]\!]$ and $[\![x; r]\!]^e$. The linearity (over the exponent) in these operations allows us to aggregate together n proofs by just having the verifier now send n random challenges and the prover responding with just a single (w, z) as before that internally adds up the respective e_i, x_i terms.

Our protocol is described in Figure 9 in Appendix F (Cramer et al.'s protocol [CDN01] is a special case). We also prove the following lemma for the protocol there:

Lemma 1. *Let $c_1, \ldots, c_n$ be Paillier ciphertexts under a public key* pk. *Assuming the security of Paillier encryption, the protocol $\Pi_{\textit{ptxt-knwldg}}$ in Figure 9 is an honest-verifier zero knowledge proof-of-knowledge of $\{(x_i, r_i)\}_{i\in[n]}$ s.t. $c_i = \mathsf{Plr.Enc}(\mathsf{pk}, x_i; r_i)$ for all i.*

8.3 L2-Norm

Finally, we present a honest-verifier zero knowledge protocol which, given n Paillier ciphertexts $[\![x_1]\!], \ldots [\![x_n]\!]$, proves that the L2 norm of vector $\overrightarrow{x} := (x_1, \ldots x_n)$ is equal to some public value y *over the integers*. The protocol is presented in Figure 10.

We use two key techniques. The first is a ZK proof that the L2 norm equals y modulo N (note that this not over integers, otherwise we would be done). This combines the use of a semi-honest multiplication protocol and an algebraic MAC to prevent a malicious prover from cheating in the following way (all operations in this protocol happens in $\mathbb{Z}_N$): First, the verifier masks and MACs each ciphertext $[\![x_i]\!]$ as $[\![w_i]\!] \leftarrow [\![x_i]\!]\alpha + \mathsf{Plr.Enc}(pk, \rho_i)$ where α, ρ_i (uniformly random in $\mathbb{Z}_N$) are the MAC key and mask respectively. This is then revealed to the prover who can compute $w_i^2 = (x_i^2\alpha^2 + 2\alpha\rho_i x_i + \rho_i^2)$. Observe that this contains the desired value x_i^2. We can isolate this value by requiring the verifier to also send $[\![v_i]\!] \leftarrow \sum_i(-2\alpha\rho_i[\![x_i]\!] - \rho_i^2) + \mathsf{Plr.Enc}(pk, \beta)$ for a mask β (uniformly random in $\mathbb{Z}_N$). The prover can then obtain a MACed version of the L2 norm as follows:

$$\begin{aligned} z &= \sum_i (x_i^2\alpha^2 + 2\alpha\rho_i x_i + \rho_i^2) + \sum_i (-2\alpha\rho_i[\![x_i]\!] - \rho_i^2 + \beta) \\ &= \sum_i x_i^2\alpha^2 + \beta \end{aligned}$$

Finally, the prover can simply send z to the verifier who checks that $\sum_i x_i^2 = y$ by checking that $z = y\alpha^2 + \beta$.

The technique above proved that the L2-norm is equal to y over $\mathbb{Z}_N$. However, we desire this equality to hold over the integers. We achieve this by adding a second instance of this protocol except that we will now logically perform the check modulo N' where $y \ll N' \ll N$ and chosen by V after the x_i are fixed.

If these two checks pass, it implies that

$$\begin{aligned} \sum_i x_i^2 = y \bmod N \quad &\implies \quad \exists t \in \mathbb{Z}^*, \sum_i x_i^2 = y + Nt \\ \sum_i x_i^2 = y \bmod N' \quad &\implies \quad \exists s \in \mathbb{Z}^*, \sum_i x_i^2 = y + N's \end{aligned}$$

Rewriting the implication we obtain $Nt = N's$. This means the prover must choose t to be a multiple of N'. However, since N' is prime and chosen after all the x_i are fixed, the prover can only do this with negligible probability.

Our protocol is formally described in Figure 10 in Appendix F. We also prove the following lemma for the protocol there:

Lemma 2. *Let $c_1 = [\![x_1]\!], \ldots, c_n = [\![x_n]\!]$ be Paillier ciphertexts under a public key* pk *and y be a positive integer. Assuming the security of Paillier encryption, $\Pi_{zk\text{-}L2}$ in Figure 10 is a honest-verifier zero knowledge protocol to show that $y = \sum_{i\in[n]} x_i^2$ (over the integers).*

Consider a more generic proof strategy where we instead prove each x_i is small therefore $\sum_i x_i^2$ can not be more than N. Even when using the optimized approximate range check of Lindell [Lin17], this strategy would require sending $O(n\lambda)$ Paillier ciphertexts. In contrast, our optimized approach only involves $O(n)$ ciphertexts. In practice this results in a speed up of at least 40 times.

9 IMPLEMENTATION

We implement our protocols and report on their performance. The implementation was written in C++ and utilizes the GMP library for Paillier operations and the oblivious transfer implementation of Chou and Orlandi [CO15].

All evaluations were performed on a single commodity laptop with an i5-8365U processor and 16GB of memory. All network traffic was routed over local host with a measured latency of 0.2 milliseconds on a 10Gbps network card. To account for network latency and bandwidth restrictions, additional running time would need to be added. The Paillier scheme is implemented using a security level analogous to 3072 bit RSA, which is estimated to achieve 112 bits of computational security. Where relevant, we use a statistical security parameter of 40 bits.

We consider various template lengths $\ell \in \{128, 256, 512, 1024\}$ combined with different bit lengths $m \in \{8, 16, 24\}$ for each template element, i.e. templates are elements in $\mathbb{Z}_{2^m}^{\ell}$. We note that this bit length is of the input size while the inner product itself can be of bit length $2m + \log_2 \ell$. We consider the three protocols Π_1 (Figure 3), Π_2 (Figure 4), Π_3 (Figure 5) as presented above where the enrollment phase for Π_2, Π_3 is separated as Π_{en}.

Π_1-Enc in Figure 8 denotes the time for $\mathcal{D}$ to locally generate an encryption of a template for Π_1 (enrollment phase), a one-time operation. Π_1 in the next column denotes the time to perform the matching phase of Π_1. We assume that the terminal $\mathcal{T}$ possesses the measurement in template form. As can be seen, this phase is significantly faster than the one-time enrollment phase. For example, encrypting a template with $\ell = 256, m = 16$ requires 0.6 seconds while the matching phase requires 0.024 seconds. This is because Paillier encryption requires computing a large exponentiation $r^N \mod N^2$ where N is 3072 bits in our case. In contrast, decryption and homomorphic operations are relatively efficient. When designing our protocols this is one of the reasons why $\mathcal{D}$ always encrypts their long term template as opposed to having $\mathcal{T}$ encrypting each measurement with its own Paillier key. Moreover, we found that simply generating the plaintext template using SphereFace [LWY+17] required more time than Π_1.

As seen in Figure 7, Π_1 has very low communication overhead. The m_1 column of Π_1 denotes the size in bytes of the first message which $\mathcal{D}$ sends to the terminal. For example, this message is of size 16KB for $\ell = 256, m = 16$. For this set of parameters we achieve a packing factor of 8×. That is, without our packing technique the encrypted template would be 128KB. An important property of our protocols is that this message can be reused in the case that several comparisons with the same $\mathcal{D}$ are performed. After $\mathcal{T}$ receives this first message, each subsequent Π_1 sessions only requires $\mathcal{T}$ sending 516 bytes, regardless of ℓ, m.

Next we consider the overhead for a possibly malicious $\mathcal{D}$ to enroll a new template (common for Π_2 and Π_3). We report the running time of this protocol in two parts. Π_{en}-Enc is the time for $\mathcal{D}$ to locally encrypt their template and $\Pi_{\mathsf{en}}$is the time required to run the (rest of) enrollment protocols which contains several zero knowledge proofs that the template is well formed. As can be seen in Figure 8, simply encrypting the template (with one element per ciphertext) requires about half the time as the rest of the enrollment protocol. This demonstrates that our custom designed

m	ℓ	Π_1		Π_2			Π_3			Π_{en}	
		m_1	$\mathcal{T} \to \mathcal{D}$	m_1	$\mathcal{D} \to \mathcal{T}$	$\mathcal{T} \to \mathcal{D}$	m_1	$\mathcal{D} \to \mathcal{T}$	$\mathcal{T} \to \mathcal{D}$	$\mathcal{D} \to \mathcal{SP}$	$\mathcal{SP} \to \mathcal{D}$
8	128	8,464	516	11,536	260	516	11,536	2,944	66,809	124,007	66,082
	256	16,656	516	22,288	260	516	22,288	2,944	69,561	189,543	131,618
	512	33,040	516	44,304	260	516	44,304	3,071	72,313	320,615	262,690
	1024	65,808	516	87,824	260	516	87,824	3,076	75,065	582,760	524,834
16	128	8,464	516	13,584	260	516	13,584	4,000	110,841	124,007	66,082
	256	16,656	516	26,896	260	516	26,896	4,000	113,593	189,543	131,618
	512	33,040	516	53,008	260	516	53,008	4,132	116,313	320,615	262,690
	1024	65,808	516	105,232	260	516	105,232	4,132	119,033	582,760	524,834
24	128	10,000	516	16,656	260	516	16,656	5,056	154,393	124,007	66,082
	256	19,216	516	33,040	260	516	33,040	5,056	157,113	189,543	131,618
	512	38,160	516	65,808	260	516	65,808	5,188	159,833	320,615	262,690
	1024	75,536	516	131,344	260	516	131,344	5,188	162,553	582,760	524,834

Figure 7: Communication overhead in bytes for various protocols. m denotes the bit count of the template elements. ℓ denotes the number of elements in the template. m_1 denotes the size of the first message of the protocol which is reusable. $\mathcal{D} \to \mathcal{T}$ denote the remaining communication from $\mathcal{D}$ to $\mathcal{T}$. $\mathcal{T} \to \mathcal{D}$ denote the total communication from $\mathcal{T}$ to $\mathcal{D}$.

m	ℓ	Π_1-Enc	Π_1	Π_{en}-Enc	Π_{en}	Π_2	Π_3
8	128	318	12	1,110	2,914	25	71
	256	626	19	2,198	5,018	44	102
	512	1,118	40	4,389	9,249	74	171
	1,024	2,263	72	8,809	17,704	138	289
16	128	457	15	1,082	2,927	33	87
	256	630	24	2,200	5,008	49	119
	512	1,131	45	4,367	9,439	89	195
	1,024	2,316	88	8,855	17,849	170	357
24	128	384	19	1,094	2,892	28	74
	256	742	32	2,204	5,053	43	105
	512	1,298	64	4,373	9,242	73	164
	1,024	2,541	118	8,772	17,822	123	266

Figure 8: Running times in milliseconds of various protocols with element bit count m and vector length ℓ.

zero knowledge protocols are quite efficient. For example, with $\ell = 256, m = 16$ encrypting the template requires 2.2 seconds while the protocol requires 5.0 seconds. We note that this protocol can trivially be parallelized but we did not consider this optimization. While the enrollment protocol requires a non-negligible amount of time, this protocol only has to be performed once during a setup phase and therefore is extremely practical. The communication overhead of this protocol is shown in Figure 7 and requires between 0.2 and 1 MB of total communication depending on ℓ, m.

For the matching phase of Π_2, we observe very practical performance. The running time of Π_2 with $\ell = 256, m = 16$ is just 0.05 seconds. The protocol consists of three messages where $\mathcal{D}$ sends the first which can be reused. For $\ell = 256, m = 16$, this first message is of size 27KB while the next per session messages are of size 516 and 260 bytes respectively. For these parameters we achieve a packing factor of 6×, reducing the first message size from 162KB. Moreover, when we consider the amortized setting where the first message is reused, this protocol requires only 0.8KB per authentication.

Finally, we consider the protocol Π_3 which strengthens Π_2 so that the inner product is not revealed to $\mathcal{T}$. This is achieved with the addition of more Paillier operations along with a small garbled circuit evaluation. In terms of running time, we observe that the protocol requires 0.12 seconds for $\ell = 256, m = 16$ where the majority of that is spent doing Paillier operations. The main overhead of Π_3 compared to Π_2 is the added communication overhead incurred by sending the garbled circuit. For $\ell = 256, m = 16$, we observe a total communication of 142KB compared to 28KB for Π_2. This difference becomes more stark in the amortized setting where the first message is reused many times. In this case Π_3 requires 116KB compared to 0.8KB for Π_2. However, given that 142KB is a very small amount communication for modern devices we argue that Π_3 achieves very practical overheads. For example, 4G wireless networks can send up to 100Mbps while the new 5G networks will be able to send up to 10Gbps. Alternatively, locally area networks such as Android Nearby are able to provide comparable high throughput communication channels.

In cases when the bandwidth of $\mathcal{D}$ is extremely constrained, we observe that a large majority of the communication overhead can be preprocessed with the assistance of $\mathcal{SP}$. In particular, using well known techniques $\mathcal{SP}$ can generate the garbled circuit and send it to $\mathcal{D}$ along with performing OT (extension) protocol on random inputs. During the online phase, $\mathcal{D}$ can instruct $\mathcal{SP}$ to send $\mathcal{T}$ the garbled circuit seed and OT messages from the preprocessing. $\mathcal{SP}$ can also send the encrypted template of $\mathcal{D}$. Then, $\mathcal{D}$ and $\mathcal{T}$ can derandomize the OTs and complete the protocol. This would result in Π_3 requiring approximately $2(2m + \log_2 n + \lambda)\kappa$ bits of communication between $\mathcal{T}$ and $\mathcal{D}$. For $\ell = 256, m = 16$, this would result in approximately 20KB of online communication. Applying the sample idea to Π_1 and Π_2 results in removing the need for $\mathcal{D}$ to send the first message because $\mathcal{SP}$ would send it on her behalf. It is easy to show that this achieves the same level of security since we already assume that $\mathcal{SP}$ and $\mathcal{T}$ can collude and are both semi-honest.

REFERENCES

[and] Android — Biometrics. https://source.android.com/security/biometric.

[appa] About Face ID advanced technology. https://support.apple.com/en-us/HT208108.

[appb] About Touch ID advanced technology. https://support.apple.com/en-us/HT204587.

[ASD+16] Paarijaat Aditya, Rijurekha Sen, Peter Druschel, Seong Joon Oh, Rodrigo Benenson, Mario Fritz, Bernt Schiele, Bobby Bhattacharjee, and

Tong Tong Wu. I-pic: A platform for privacy-compliant image capture. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, page 235–248, New York, NY, USA, 2016. Association for Computing Machinery.

[BG11] Marina Blanton and Paolo Gasti. Secure and efficient protocols for iris and fingerprint identification. In Vijay Atluri and Claudia Díaz, editors, *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, volume 6879 of *Lecture Notes in Computer Science*, pages 190–209. Springer, 2011.

[bio19] Major breach found in biometrics system used by banks, UK police and defence firms. https://www.theguardian.com/technology/2019/aug/14/major-breach-found-in-biometrics-system-used-by-banks-uk-police-and-defence-firms, August 2019.

[Boy04] Xavier Boyen. Reusable cryptographic fuzzy extractors. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, page 82–91, New York, NY, USA, 2004. Association for Computing Machinery.

[cbp19] U.S. Customs and Border Protection says photos of travelers were taken in a data breach. https://www.washingtonpost.com/technology/2019/06/10/us-customs-border-protection-says-photos-travelers-into-out-country-were-recently-taken-data-breach/, June 2019.

[CDN01] Ronald Cramer, Ivan Damgα rd, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–299. Springer, 2001.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, volume 9230 of *Lecture Notes in Computer Science*, pages 40–58. Springer, 2015.

[CO17] Wutichai Chongchitmate and Rafail Ostrovsky. Circuit-private multi-key FHE. In Serge Fehr, editor, *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part II*, volume 10175 of *Lecture Notes in Computer Science*, pages 241–270. Springer, 2017.

[CRC+19] Rahul Chatterjee, M. Sadegh Riazi, Tanmoy Chowdhury, Emanuela Marasco, Farinaz Koushanfar, and Ari Juels. Multisketches: Practical secure sketches using off-the-shelf biometric matching algorithms. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1171–1186, New York, NY, USA, 2019. Association for Computing Machinery.

[DGK07] Ivan Damgα rd, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *Information Security and Privacy*, pages 416–430, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540. Springer, 2004.

[EFG+09] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In Ian Goldberg and Mikhail J. Atallah, editors, *Privacy Enhancing Technologies, 9th International Symposium, PETS 2009, Seattle, WA, USA, August 5-7, 2009. Proceedings*, volume 5672 of *Lecture Notes in Computer Science*, pages 235–253. Springer, 2009.

[goo] Nearby Connections API. https://developers.google.com/nearby/connections/overview.

[HMEK11] Yan Huang, Lior Malka, David Evans, and Jonathan Katz. Efficient privacy-preserving biometric identification. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*. The Internet Society, 2011.

[HMR+19] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient RSA key generation and threshold paillier in the two-party setting. *J. Cryptology*, 32(2):265–323, 2019.

[HMRT12] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA key generation and threshold paillier in the two-party setting. In Orr Dunkelman, editor, *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*, volume 7178 of *Lecture Notes in Computer Science*, pages 313–331. Springer, 2012.

[jet19] Passenger Questions JetBlue Facial-Recognition System. https://www.wbur.org/hereandnow/2019/05/02/jetblue-facial-recognition-check-in, May 2019.

[KAMR04] Florian Kerschbaum, Mikhail J. Atallah, David M'Raïhi, and John R. Rice. Private fingerprint verification without local storage. In David Zhang and Anil K. Jain, editors, *Biometric Authentication, First International Conference, ICBA 2004, Hong Kong, China, July 15-17, 2004, Proceedings*, volume 3072 of *Lecture Notes in Computer Science*, pages 387–394. Springer, 2004.

[KSvdV+05] Tom A. M. Kevenaar, Geert Jan Schrijen, Michiel van der Veen, Anton H. M. Akkermans, and Fei Zuo. Face recognition with renewable and privacy preserving binary templates. In *Proceedings of the Fourth IEEE Workshop on Automatic Identification Advanced Technologies (AutoID 2005), 16-18 October 2005, Buffalo, NY, USA*, pages 21–26. IEEE Computer Society, 2005.

[Lin17] Yehuda Lindell. Fast secure two-party ECDSA signing. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 613–644. Springer, 2017.

[LWY+17] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 6738–6746. IEEE Computer Society, 2017.

[OPJM10] Margarita Osadchy, Benny Pinkas, Ayman Jarrous, and Boaz Moskovich. Scifi - A system for secure face identification. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 239–254. IEEE Computer Society, 2010.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[sam] Samsung Pass. https://www.samsung.com/global/galaxy/apps/samsung-pass/.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015*, pages 815–823. IEEE Computer Society, 2015.

[SSW09] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Efficient privacy-preserving face recognition. In Dong Hoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers*, volume 5984 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2009.

[ST07] Berry Schoenmakers and Pim Tuyls. Computationally secure authentication with noisy data. In *Security with Noisy Data*, pages 141–149. Springer London, 2007.

[TP91] Matthew A. Turk and Alex Pentland. Face recognition using eigenfaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 1991, 3-6 June, 1991, Lahaina, Maui, Hawaii, USA*, pages 586–591. IEEE, 1991.

[WWZ+18] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 5265–5274. IEEE Computer Society, 2018.

A RELATED WORK

A.1 Authentication vs. Identification

Biometrics can be used to authenticate or identify individuals. Authentication is a $1:1$ type of task where a measurement is compared against a template to determine if they belong to the same person. Identification, on the other hand, is a $1:N$ type of task where a measurement is matched against a potentially large set of N templates to determine who the measurement belongs to. The use of biometrics on mobile devices like smartphones falls into the first category, whereas their use for law enforcement, civil identity, border control, etc. generally falls into the second category.

Identification tasks require large biometric databases which become very attractive targets for hackers, leading to massive breaches [bio19, cbp19]. Consumers are also very concerned about their privacy [jet19]. Authentication on mobile devices, as discussed before, is local in nature. Biometric readings and the derived templates never leave the user device [appb, appa, and, sam].

The kind of *external-facing* authentication discussed here is essentially an authentication task, but communication with an external sensor could leak sensitive information (which we intend to prevent). One may observe some similarities with identification too because a measurement taken by the external sensor could be matched against several devices to find the right person. However, this should be seen as multiple instances of authentication rather than an instance of identification.

A.2 Detailed Look

The first paper to consider comparing biometric data within MPC is that of Kerschbaum et al. [KAMR04]. This protocol first extracts the locations of so-called minutia points[4] from each biometric measurement and then use an MPC protocol to determine if they have more than a threshold number of minutia points in common. This is achieved by dividing the minutia space into a grid. Each biometric is mapped to a n bit vector v where $v_i = 1$ if and only if there is a minutia point in grid square i. Using additive homomorphic encryption, the number of matches can then computed as $n - \mathsf{HD}(v, v')/2$, where HD denotes Hamming distance, which is compared with some threshold t. Schoenmakers and Tuyls [ST07] perform a similar computation but where an iris scan is the biometric measurement.

Erkin et al. [EFG+09] considered a setting where a client has a photo P of their face and a server has a database of photos $I = (I_1, \ldots, I_m)$. This work uses the Eigenface framework [TP91] where the server generates a model M using the dataset I. They consider this model to be the private input of the server. Using an additive homomorphic encryption of P, the parties compute the linear circuit $[\![\Omega]\!] := \mathsf{Embed}([\![P]\!], M) \in \mathbb{Z}^n$. Given this, the distance metric between P, I_i can then be computed as the L2-norm $[\![d_i]\!] := ||[\![\Omega]\!], \mathsf{Embed}(I_i, M)||$. Finally, using an interactive protocol, the i with the minimum distance d_i which is less than some threshold t is revealed to the client. If no such i exists, a special value is revealed.

Sadeghi et al. [SSW09] gave several improvements to this protocol. First, they replace the final phase where the minimum distance is computed with a garbled circuit protocol. They show that this approach is significantly faster due to Paillier additive homomorphic encryption being poorly suited for binary circuits. Secondly, they show that the m computations of the L2-norms can be improved using a technique known as batching. For security reasons Paillier ciphertext must have a large plaintext space with more than 1024-bits while the values encrypted in the Erkin et al. protocol [EFG+09] can be 32-bits or less. As such, Sadeghi et al. showed that many $\mathsf{Embed}(I_i, M)$ vectors can be batched together into the same set of ciphertexts. In this way, several distance functions can be computed for each L2-norm computation applied to $[\![\Omega]\!]$.

[4] An example of a minutia point is the location on a figure print where two ridges merger together.

Haung et al. [HMEK11] further optimize this protocol in various ways. First they assume that the model M is public which allows the client to encrypt $[\![\Omega]\!]$. This greatly reduces the number of homomorphic operations. Secondly, they optimized for the amortized setting where the client wants to authenticate many measurements with the server. In addition, they further optimize the computation of the minimum distance (i, d_i) by first computing d_i and then a separate protocol to retrieve the index i and any associated value.

Osadchy et al. [OPJM10] present the SCiFI protocol which introduces a new facial recognition algorithm intended to be more efficient to compute when performed with MPC. First their system has some party generate a model M from some dataset D. This model is then revealed to all parties. The model contains many sets of cropped images. Each set C_i of images is centered at some location L_i on the face, e.g., the left eye. The images in each set C_i have a similar look where this is determined using some distance metric. To compare two images P_1, P_2, each is turned into a set of indices S_1, S_2 where S_j contains index i if the cropped region of P_j centered at L_i looks similar to the images in C_i. If the size of the intersection between S_1, S_2 is larger than some threshold t, then it is considered a match. This size is computed in the same way as Kerschbaum et al. [KAMR04] which is to turn each set into a vector v, v' and compute $n - \mathsf{HD}(v, v')/2$ using additive homomorphic encryption.

A.3 Comparison with LWE-based AHE

One could also consider implementing our protocol using LWE-based additive homomorphic encryption. These schemes typically support the packing of several values into a ciphertext such that the ciphertext can be operated on as a vector of elements. However, our choice of Paillier results in less communication which is a key performance metric. LWE ciphertexts typically have to be large due to security consideration. For example, with a vector length of $n = 256$ and $m = 16$-bit features the resulting communication for protocol Π_1 is about 17KB while this implemented using LWE results in 96KB. However, we do note that the running time of the LWE solution is faster at just 10 milliseconds compared to our protocol requiring 45 milliseconds. The LWE parameters in question also supports up to $n = 2048$ length vectors with no added overhead. This means that if the input vectors are of sufficient length then the LWE approach could give better communication and running time. The final advantage of our approach is that Paillier more easily translates to the malicious setting. ZK-proofs of LWE ciphertexts add additional overhead which could impact the practicality of Π_2, Π_3. We leave a more detailed comparison of LWE and Paillier in the context of our protocols for future work.

B PRELIMINARIES

In this section, we formally define some of the primitives we use.

B.1 Paillier Encryption

The additively homomorphic encryption scheme by Paillier [Pai99] (Plr.Setup, Plr.Enc, Plr.Add, Plr.Dec) satisfies the following correctness and security properties. We refer to [Pai99] for the proofs.

- **Correctness.** For any $(\mathsf{epk}, \mathsf{esk}) \leftarrow \mathsf{Plr.Setup}(\lambda)$:

(1) For any message x, we have with overwhelmingly large probability

$$\mathsf{Plr.Dec}(\mathsf{esk}, \mathsf{Plr.Enc}(\mathsf{epk}, \mathsf{x})) = \mathsf{x}.$$

(2) For any two messages x_0 and x_1 with $[\![x_b]\!] = \mathsf{Plr.Enc}(\mathsf{epk}, \mathsf{x}_b)$ for $b \in \{0,1\}$, we have

$$\mathsf{Plr.Dec}(\mathsf{esk}, \mathsf{Plr.Add}([\![x_0]\!], [\![x_1]\!])) = \mathsf{x}_0 + \mathsf{x}_1.$$

- **Semantic Security.** For all PPT algorithms $\mathcal{A}$ and any two messages x_0 and x_1, we have

$$|\Pr[\mathcal{A}(\mathsf{epk}, (\mathsf{x}_0, \mathsf{x}_1), [\![x_0]\!]) = 1] - \Pr[\mathcal{A}(\mathsf{epk}, (\mathsf{x}_0, \mathsf{x}_1), [\![x_1]\!]) = 1]| \leq negl(\lambda),$$

where $(\mathsf{epk}, \mathsf{esk}) = \mathsf{Plr.Setup}(\lambda)$ and $[\![x_b]\!] = \mathsf{Plr.Enc}(\mathsf{epk}, \mathsf{x}_b)$ for $b \in \{0,1\}$.

- **Circuit Privacy.** Informally, circuit privacy requires that the ciphertext generated by a set of homomorphic operations does not reveal anything about the (linear) circuit that it evaluates, beyond the output value of that circuit, even to an adversary who generated the public and secret keys. We refer to [CO17] for a formal definition and note that this property is easily achieved by the Paillier encryption scheme.

B.2 Zero Knowledge Proofs

An interactive protocol between a prover P and a verifier V for deciding a language L is an honest-verifier zero-knowledge proof of knowledge protocol if it satisfies the following properties. Below, we define $\langle P(\cdot), V(\cdot)\rangle$ to be the output of $V(\cdot)$ at the end of the protocol execution.

- **Completeness.** For every security parameter $\lambda \in \mathbb{N}$, and any $(x, w) \in R_L$,

$$\Pr[\langle P(x, w), V(x)\rangle = 1] = 1 - negl(\lambda).$$

where the probability is over the randomness of P and V.

- **Soundness.** For any PPT cheating prover P^*, it holds that if $x^* \notin L$ then

$$\Pr[\langle P^*(x^*), V(x^*)\rangle = 1] = negl(\lambda).$$

where the probability is over the random coins of V.

- **Proof of Knowledge.** For any PPT cheating prover P^*, there exists a PPT extractor $\mathsf{ZK.Ext}$ such that for all x : $\Pr[\langle P^*(x^*), V(x^*)\rangle = 1 \wedge (w \leftarrow \mathsf{ZK.Ext}^{P^*}(x^*) \text{ s.t } (x^*, w^*) \notin R_L)] = negl(\lambda)$. where the probability is over the random coins of V and $\mathsf{ZK.Ext}$.

- **Zero Knowledge.** For any (semi-) honest verifier V^*, there exists a PPT simulator Sim such that for all $x \in L$: the view of V^* when interacting with an honest prover P on input (x, w) is computationally indistinguishable from its view when interacting with Sim on input x alone.

B.3 Digital Signatures

A digital signature scheme consists of the following three algorithms (Gen, Sign, Verify).

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$. A randomized algorithm that takes the security parameter λ as input, and generates a verification-key vk and a signing key sk.
- $\mathsf{Sign}(\mathsf{sk}, m) =: \sigma$. A randomized algorithm that takes a mesage m and signing key sk as input and outputs a signature σ.
- $\mathsf{Verify}(\mathsf{vk}, (m, \sigma)) =: 1/0$. A deterministic algorithm that takes a verification key vk and a candidate message-signature pair (m, σ) as input, and outputs 1 for a valid signature and 0 otherwise.

The following correctness and security properties should be satisfied:

- **Correctness.** For all $\lambda \in \mathbb{N}$, all $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, any message m, $\mathsf{Verify}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$.
- **Unforgeability** A signature scheme is unforgeable if for any PPT adversary $\mathcal{A}$, the following game outputs 1 with negligible probability (in security parameter).
  - *Initialize.* Run $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Give vk to $\mathcal{A}$. Initiate a list $L := \emptyset$.
  - *Signing queries.* On query m, return $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$. Run this step as many times as $\mathcal{A}$ desires. Then, insert m into the list L.
  - *Output.* Receive output $(m^\star, \sigma^\star)$ from $\mathcal{A}$. Return 1 if and only if $\mathsf{Verify}(\mathsf{vk}, (m^\star, \sigma^\star)) = 1$ and $m^\star \notin L$, and 0 otherwise.

B.4 Two-Message Oblivious Transfer

A two-message oblivious transfer (OT) protocol is a tuple ($\mathsf{OT.Round}_1$, $\mathsf{OT.Round}_2$, $\mathsf{OT.Output}$) defined as follows:

- $\mathsf{OT.Round}_1(\beta)$: A PPT algorithm that, given a bit $\beta \in \{0,1\}$, outputs a message m_1 and a secret state st.
- $\mathsf{OT.Round}_2((\mu_0, \mu_1), \mathsf{m}_1)$: A PPT algorithm that, given a pair of strings $(\mu_0, \mu_1) \in \{0,1\}^{2\lambda}$ and a message m_1, outputs a message m_2.
- $\mathsf{OT.Output}(\mathsf{st}, \beta, \mathsf{m}_2)$: A deterministic algorithm that, given a secret state st, a bit $\beta \in \{0,1\}$ and a message m_2, outputs a string $\mu' \in \{0,1\}^\lambda$.

The following correctness and security properties should be satisfied:

- **Correctness.** For any $\lambda \in \mathbb{N}$, any bit $\beta \in \{0,1\}$, any pair of strings $(\mu_0, \mu_1) \in \{0,1\}^{2\lambda}$: let $(\mathsf{m}_1, \mathsf{st}) = \mathsf{OT.Round}_1(\beta)$, $\mathsf{m}_2 = \mathsf{OT.Round}_2((\mu_0, \mu_1), \mathsf{msg}_1)$, $\mu' = \mathsf{OT.Output}(\mathsf{st}, \beta, \mathsf{m}_2)$. We have $\mu' = \mu_\beta$ with overwhelmingly large probability in λ.
- **Receiver Privacy.** For any $\lambda \in \mathbb{N}$, for any PPT adversary $\mathcal{A}$: we have $\Pr[\mathcal{A}(\mathsf{OT.Round}_1(0)) = 0] - \Pr[\mathcal{A}(, \mathsf{OT.Round}_1(1) = 0] \leq negl(\lambda)$, where the probability is defined over the random coins of $\mathsf{OT.Round}_1$ algorithm.
- **Sender Privacy**. For any $\lambda \in \mathbb{N}$, any $\beta \in \{0,1\}$, any $(\mu_0, \mu_1) \in \{0,1\}^{2\lambda}$, any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathsf{OT.Round}_2.\mathsf{Sim}$ such that the following holds: Let $\mathsf{m}_1 \leftarrow \mathcal{A}(\beta)$. Then,

$$|\Pr[\mathcal{A}(\beta, \mathsf{m}_2) = 0] - \Pr[\mathcal{A}(\beta, \widehat{\mathsf{m}_2}) = 0]| \leq negl(\lambda),$$

where $\mathsf{m}_2 = \mathsf{OT.Round}_2((\mu_0, \mu_1), \mathsf{m}_1)$,
$\widehat{\mathsf{m}_2} = \mathsf{OT.Round}_2.\mathsf{Sim}(\mu_\beta, \mathsf{m}_1)$.

The protocol can be in the CRS model (in which case we would also have an algorithm $\mathsf{OT.Setup}(\cdot)$ to generate the CRS) or in the Random oracle model.

B.5 Garbled Circuits

A garbling scheme for a class of circuits C with n-bit inputs consists of the following polynomial-time algorithms:

- $\mathsf{Garble}(C \in \mathcal{C})$: A probabilistic algorithm that takes a circuit $C \in \mathcal{C}$ as input and outputs a garbled circuit $\widetilde{C}$ and a set of labels $\overrightarrow{\ell} = \{\ell_{j,0}, \ell_{j,1}\}_{j\in[n]}$.
- $\mathsf{Eval}((x_1, \ldots, x_n) \in \{0,1\}^n, \{\ell_{j,x_j}\}_{j\in[n]}, \widetilde{C})$: A deterministic algorithm that takes as input a string $(x_1, \ldots, x_n) \in \{0,1\}^n$, a set of labels $\{\ell_{j,x_j}\}_{j\in[n]}$ and a garbled circuit $\widetilde{C}$, and outputs a bit $y \in \{0,1\}$.

The following correctness and security properties should be satisfied:

- **Correctness.** For any circuit $C \in \mathcal{C}$, any string $(x_1, \ldots, x_n) \in \{0,1\}^n$, let $(\widetilde{C}, \{\ell_{j,0}, \ell_{j,1}\}_{j\in[n]}) = \mathsf{Garble}(C)$. We have
$$\mathsf{Eval}((x_1, \ldots, x_n), \{\ell_{j,x_j}\}_{j\in[n]}, \widetilde{C}) = C(x_1, \ldots, x_n).$$
- **Security.** There exists a PPT algorithm $\mathsf{Garble.Sim}(\cdot)$ such that for any circuit $C \in \mathcal{C}$, any string $(x_1, \ldots, x_n) \in \{0,1\}^n$, the ensembles
$$(\widetilde{C}, \{\ell_{j,x_j}\}_{j\in[n]}) \quad \text{and} \quad \mathsf{Garble.Sim}(1^\lambda, C(x))$$
are computationally indistinguishable where $(\widetilde{C}, \{\ell_{j,x_j}\}_{j\in[n]}) \leftarrow \mathsf{Garble}(C)$.

C PROTOCOL 1: PROOF

C.1 Corrupt Device

Consider a semi-honest adversary $\mathcal{A}$ that corrupts the device $\mathcal{D}$. We now describe the simulator Sim. Let Mask.Sim denote the simulator for the packing scheme.

Enrollment Phase. Before the phase begins, Sim gets as input $\overrightarrow{u} = (u_1, \ldots, u_\ell)$ - the input vector of the adversarial device. Upon receiving $(\mathsf{epk}, \mathsf{esk}, \{[\![u_i^*]\!]\}_{i\in[t]})$ from $\mathcal{A}$, Sim sends message "Enrolled".

Matching Phase: Before the phase begins, Sim gets the output bit y and the leakage $L_{\mathsf{dev}}(\cdot) = \mathsf{IP}$ from the ideal functionality. Sim does the following:

- **Round 1:** Receive $(\mathsf{epk}, \{[\![u_i^*]\!]\}_{i\in[t]})$ from $\mathcal{A}$.
- **Round 2:**
  (1) Pick n random strings $\mathsf{IP}_1, \ldots, \mathsf{IP}_n$ such that $(\mathsf{IP}_1 + \ldots + \mathsf{IP}_n) = \mathsf{IP}$.
  (2) Compute and send $[\![\widetilde{\mathsf{IP}}]\!] = [\![\mathsf{Mask.Sim}(\mathsf{IP}_1, \ldots, \mathsf{IP}_n)]\!]$ to $\mathcal{A}$.

We show that the above simulation strategy is successful via a series of computationally indistinguishable hybrids where the first hybrid Hyb_0 corresponds to the real world and the last hybrid Hyb_2 corresponds to the ideal world.

(1) Hyb_0**: Real world.** In this hybrid, consider a simulator SimHyb that plays the role of the honest service provider and terminal as in the real world.
(2) Hyb_1**: Switch Ciphertexts.** In this hybrid, SimHyb runs the matching phase as follows:
- Pick n random strings $\mathsf{IP}_1, \ldots, \mathsf{IP}_n$ such that $(\mathsf{IP}_1 + \ldots + \mathsf{IP}_n) = \mathsf{IP}$.
- Compute $[\![\widetilde{z}]\!]$ using the algorithm Plr.Enc where $\widetilde{z} = \mathsf{Encode}_2(\mathsf{IP}_1, \ldots, \mathsf{IP}_n)$.
- Pick n random strings $\rho_1, \ldots, \rho_n$ such that $(\rho_1 + \ldots + \rho_n) = 0$ and $\forall i, \rho_i \in \mathbb{Z}_{2^k}$. Let $\widetilde{\rho} = \mathsf{Encode}_2(\rho_1, \ldots, \rho_n)$.
- Compute and send $[\![\widetilde{\mathsf{IP}}]\!] = [\![\mathsf{Mask}(\widetilde{z} + \widetilde{\rho})]\!]$ to $\mathcal{D}$.

(3) Hyb_2**: Masking Simulator.** In this hybrid, SimHyb runs the simulator Mask.Sim of the packing scheme to compute $[\![\widetilde{\mathsf{IP}}]\!]$ in round 2 instead of running the honest masking algorithm. This hybrid corresponds to the ideal world.

We now show that every pair of successive hybrids is computationally indistinguishable.

Lemma 3. *Assuming the circuit privacy of Paillier encryption,* Hyb_0 *is computationally indistinguishable from* Hyb_1.

Proof. The only difference between the adversary's view in both worlds is the way the ciphertext $[\![\widetilde{z}]\!]$ is computed. In the real world, $\mathcal{T}$ computes $[\![\widetilde{z}]\!]$ by performing the homomorphic operations using its input vector $\overrightarrow{w}$ while in the ideal world, Sim uses random $(\mathsf{IP}_1, \ldots, \mathsf{IP}_n)$ such that $(\mathsf{IP}_1 + \ldots + \mathsf{IP}_n) = \mathsf{IP}$ and sets $\widetilde{z} = \mathsf{Encode}_2(\mathsf{IP}_1, \ldots, \mathsf{IP}_n)$. Since the ρ_i values all add up to 0, observe that in both cases, $\mathsf{Decode}(\mathsf{Plr.Dec}(\mathsf{esk}, [\![\mathsf{IP}^*]\!])$ produces a tuple $(\mathsf{IP}_1, \ldots, \mathsf{IP}_n)$ such that $\sum_{i=1}^n \mathsf{IP}_i = \mathsf{IP}$. Therefore, if there exists an adversary $\mathcal{A}$ that can distinguish between the two worlds with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{Plr}}$ that can break the circuit privacy of the Paillier encryption scheme which is a contradiction. □

Lemma 4. *Assuming the security of the packing scheme,* Hyb_1 *is statistically indistinguishable from* Hyb_2.

Proof. The only difference between the two hybrids is that in Hyb_1, SimHyb computes the ciphertext $[\![\widetilde{\mathsf{IP}}]\!]$ by homomorphically evaluating the honest masking algorithm Mask while in Hyb_2, SimHyb homomorphically computes the encoding output by the simulator Mask.Sim. From the description of the simulator Mask.Sim in Section 4.2, we crucially note that the operations (after sampling the randomness) can be performed homomorphically using Paillier encryption. Thus, from the security property of the packing scheme, the two hybrids are statistically indistinguishable. □

C.2 Corrupt Provider, Terminal

Consider a semi-honest adversary $\mathcal{A}$ that jointly corrupts the service provider $\mathcal{SP}$ and terminal $\mathcal{T}$. We now describe the simulator Sim.

Enrollment Phase. Sim sets vector $\overrightarrow{u} = (u_1, \ldots, u_\ell)$ uniformly at random and interacts with $\mathcal{A}$ in the Enrollment phase as done by $\mathcal{D}$ in the real world. That is:

(1) Compute $u_1^*, \ldots, u_t^*$ as the encoding of $\overrightarrow{u}$.
(2) Compute $(\mathsf{esk}, \mathsf{epk}) \leftarrow \mathsf{Plr.Setup}(1^\lambda)$. $\forall i \in [t]$, compute $[\![u_i^*]\!] = \mathsf{Plr.Enc}(\mathsf{epk}, u_i^*)$.
(3) Send $(\mathsf{epk}, \{[\![u_i^*]\!]\}_{i\in[t]})$ to $\mathcal{A}$.

Matching Phase: Before the phase begins, Sim gets input $(\overrightarrow{w}, \rho, y)$ where $\overrightarrow{w}$ is $\mathcal{A}$'s input, ρ is $\mathcal{A}$'s randomness, y is the output bit. Sim does the following:

- **Round 1:** Send $(\mathsf{epk}, \{[\![u_i^*]\!]\}_{i\in[t]})$ to $\mathcal{A}$.

- **Round 2:** Receive $[\![\mathsf{IP}^*]\!]$ from $\mathcal{A}$.
- **Round 3:** Send bit y to $\mathcal{A}$.

We now show that the above simulation strategy is successful.

Lemma 5. *Assuming the semantic security of Paillier encryption, the real world is computationally indistinguishable from the ideal world.*

Proof. Observe that the only difference between the two worlds is the way the ciphertexts $\{[\![u_i^*]\!]\}_{i\in[t]}$ are computed. In the real world, $\mathcal{D}$ computes them using its input vector $\overrightarrow{u}$ while in the ideal world Sim computes them as an encryption of a uniformly random vector. The adversary $\mathcal{A}$ only gets the public key epk and does not learn the secret key esk of the encryption scheme. Therefore, if there exists an adversary $\mathcal{A}$ that can distinguish between the two worlds with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{Plr}}$ that can break the semantic security of the Paillier encryption scheme which is a contradiction. □

D PROTOCOL 2: PROOF

D.1 Corrupt Device

Consider a malicious adversary $\mathcal{A}$ that corrupts the device $\mathcal{D}$. We now describe the simulator Sim. Let ZK.Ext denote the extractor for the zero knowledge protocol $\Pi_{\text{ptxt-knwldg}}$. Let Mask.Sim denote the simulator for the packing scheme.

Enrollment Phase. Sim does the following.

- Receive (epk, $[\![u_1]\!]$, ..., $[\![u_\ell]\!]$) from $\mathcal{A}$.
- Run zero knowledge protocols Π_{epk}, $\Pi_{\text{ptxt-knwldg}}$ and $\Pi_{\text{zk-L2}}$ as the verifier with $\mathcal{A}$ as the prover with statement epk for the first proof and statement (epk, $[\![u_1]\!]$, ..., $[\![u_\ell]\!]$) for the next two proofs. Also, run the extractor ZK.Ext on protocol $\Pi_{\text{ptxt-knwldg}}$ to extract witness $\{u_i, \rho_i\}_{i=1}^{\ell}$.
- If either of the proofs fail or the extractor ZK.Ext fails, send the message "Failed" to $\mathcal{A}$ and query the ideal functionality $\mathcal{F}$ as part of enrollment with a vector of norm larger than ζ, thereby instructing it to deliver the message "Failed" to the honest service provider.
- Else, do the following:
  (1) Generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda)$.
  (2) Compute $([\![u_1^*]\!], \ldots, [\![u_t^*]\!])$ using Plr.Add$(\cdot)$ where $u_1^*, \ldots, u_t^*$ is the level-1 encoding of $\overrightarrow{u} = (u_1, \ldots, u_\ell)$.
  (3) Send $\sigma = \mathsf{Sign}(\mathsf{sk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!]))$ to $\mathcal{A}$.
  (4) Query the ideal functionality $\mathcal{F}$ as part of enrollment with vector $\overrightarrow{u}$ extracted from ZK.Ext.

Matching Phase: Sim does the following.

- **Output from ideal functionality $\mathcal{F}$:** Query $\mathcal{F}$ with a new sub-session for the matching phase to receive output bit y.
- **Round 1:** Receive $(\sigma, \{[\![u_i^*]\!]\}_{i\in[t]})$ from $\mathcal{A}$.
- **Round 2:**
  (1) Output $\perp$ if $\mathsf{Verify}(\mathsf{vk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!]), \sigma) \neq 1$.
  (2) Output "Special Abort" if $\mathsf{Verify}(\mathsf{vk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!]), \sigma) = 1$ (AND) $([\![u_1^*]\!], \ldots, [\![u_t^*]\!], \sigma)$ is not the tuple sent to $\mathcal{A}$ in the Enrollment phase.
  (3) Pick a random value IP satisfying the following: $\mathsf{IP} < T$ if $y = 0$ and $T \leq \mathsf{IP} \leq \mathsf{IP}_{\mathsf{Max}}$ if $y = 1$.
  (4) Pick random strings $a \in \mathbb{Z}_\lambda, b \in \mathbb{Z}_{(\lambda \cdot \log \mathsf{IP}_{\mathsf{Max}} + \lambda)}$.
  (5) Pick n random strings $X_1, \ldots, X_n$ such that $(X_1 + \ldots + X_n) = (a \cdot \mathsf{IP} + b)$.
  (6) Compute and send $[\![\widetilde{X}]\!] = [\![\mathsf{Mask.Sim}(X_1, \ldots, X_n)]\!]$ to $\mathcal{A}$.
- **Round 3:** Receive X from $\mathcal{A}$.
- **Round 4:**
  (1) Set $\mathsf{IP}' = \frac{X-b}{a}$.
  (2) If $\mathsf{IP}_{\mathsf{Max}} < \mathsf{IP}'$, instruct $\mathcal{F}$ to output $\perp$ to the honest terminal.
  (3) If $T \leq \mathsf{IP} \leq \mathsf{IP}_{\mathsf{Max}}$, send 1 to $\mathcal{A}$. Else, send 0. Instruct $\mathcal{F}$ to deliver output to the honest terminal.

We show that the above simulation strategy is successful via a series of computationally indistinguishable hybrids where the first hybrid Hyb_0 corresponds to the real world and the last hybrid Hyb_5 corresponds to the ideal world.

(1) Hyb_0**: Real world.** In this hybrid, consider a simulator SimHyb that plays the role of the honest service provider and terminal as in the real world.
(2) Hyb_1**: Run the Extractor.** In this hybrid, SimHyb runs the extractor ZK.Ext for the zero knowledge protocol $\Pi_{\text{ptxt-knwldg}}$ as done by Sim in the ideal world to extract witness $\overrightarrow{u} = (u_1, \ldots, u_\ell)$. It also queries the ideal functionality in the enrollment phase as done by Sim.
(3) Hyb_2**: Special Abort for signature forgery.** In this hybrid, in round 2 of the matching phase, SimHyb outputs "Special Abort" if $\mathsf{Verify}(\mathsf{vk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!]), \sigma) = 1$ (AND) $(\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!], \sigma)$ is not the tuple sent to $\mathcal{A}$ in the Enrollment phase, as done by Sim in the ideal world.
(4) Hyb_3**: Switch Ciphertexts.** In this hybrid, SimHyb computes the ciphertexts in the matching phase as follows:
  - Pick a random value IP satisfying the following: $\mathsf{IP} < T$ if $y = 0$ and $T \leq \mathsf{IP} \leq \mathsf{IP}_{\mathsf{Max}}$ if $y = 1$.
  - Pick random strings $a \in \mathbb{Z}_\lambda, b \in \mathbb{Z}_{(\lambda \cdot \log \mathsf{IP}_{\mathsf{Max}} + \lambda)}$.
  - Pick n random strings $X_1, \ldots, X_n$ such that $(X_1 + \ldots + X_n) = (a \cdot \mathsf{IP} + b)$.
  - Compute $[\![\widetilde{\mathsf{z}}]\!]$ using algorithm Plr.Enc where $\widetilde{\mathsf{z}} = \mathsf{Encode}_2(X_1, \ldots, X_n)$.
  - Pick n random strings $\rho_1, \ldots, \rho_n$ such that $(\rho_1 + \ldots + \rho_n) = 0$ and $\forall i, \rho_i \in \mathbb{Z}_{2^k}$. Let $\widetilde{\rho} = \mathsf{Encode}_2(\rho_1, \ldots, \rho_n)$.
  - Compute and send $[\![\widetilde{X}]\!] = [\![\mathsf{Mask}(\widetilde{\mathsf{z}} + \widetilde{\rho})]\!]$ to $\mathcal{D}$.
(5) Hyb_4**: Masking Simulator.** In this hybrid, SimHyb runs the simulator Mask.Sim of the packing scheme to compute $[\![\widetilde{X}]\!]$ in round 2 instead of running the honest masking algorithm.
(6) Hyb_5**: Ideal world.** In this hybrid, SimHyb instructs the ideal functionality to either deliver output $\perp$ or the real output to the honest terminal as done by Sim in round 4. This hybrid corresponds to the ideal world.

We now show that every pair of successive hybrids is computationally indistinguishable.

Lemma 6. *Assuming the proof of knowledge property for protocol $\Pi_{\text{ptxt-knwldg}}$ and soundness of protocols $\Pi_{\text{epk}}, \Pi_{\text{zk-L2}}$, Hyb_0 is computationally indistinguishable from Hyb_1.*

Proof. The difference between the two hybrids is that in Hyb_1, SimHyb also runs the extractor NIZK.Ext on the proofs given by the adversary for protocol $\Pi_{\text{ptxt-knwldg}}$ to compute its input vector

$\overrightarrow{u}$ and checks that the L2-norm of $\overrightarrow{u}$ is ζ. From the soundness of protocols $\Pi_{\mathsf{epk}}, \Pi_{\mathsf{zk\text{-}L2}}$, we know that if the proofs verify successfully, it must indeed be the case that the public key was correctly generated and that the L2-norm of the underlying vector is ζ if the ciphertexts were correctly generated. Thus, the only difference between the two hybrids is if the adversary can produce a proof for protocol $\Pi_{\mathsf{ptxt\text{-}knwldg}}$ such that, with non-negligible probability, the proof verifies successfully, but SimHyb fails to extract $\overrightarrow{u}$ and hence SimHyb aborts. However, we can show that if there exists an adversary $\mathcal{A}$ that can cause this to happen with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{ZK}}$ that breaks the argument of knowledge property of protocol $\Pi_{\mathsf{ptxt\text{-}knwldg}}$ with non-negligible probability which is a contradiction. □

Lemma 7. *Assuming the unforgeability of the signature scheme,* Hyb_1 *is computationally indistinguishable from* Hyb_2.

Proof. The difference between the two hybrids is that in Hyb_2, in round 2 of the matching phase, SimHyb outputs "Special Abort". Observe that this happens if in round 1, $\mathcal{A}$ sends a tuple $(\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!], \sigma)$ such that this was not the tuple sent to $\mathcal{A}$ in the enrollment phase and $\mathsf{Verify}(\mathsf{vk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!]), \sigma) = 1$. However, if there exists an adversary $\mathcal{A}$ that can cause this event to happen with non-negligible probability, we can use $\mathcal{A}$ to design a reduction $\mathcal{A}_{\mathsf{Sign}}$ that breaks the unforgeability of the signature scheme with non-negligible probability which is a contradiction. □

Lemma 8. *Assuming the circuit privacy of Paillier encryption,* Hyb_2 *is computationally indistinguishable from* Hyb_3.

Proof. This is identical to the proof of Lemma 3. □

Lemma 9. *Assuming the security of the packing scheme,* Hyb_3 *is statistically indistinguishable from* Hyb_4.

Proof. This is identical to the proof of Lemma 4. □

Lemma 10. Hyb_4 *is statistically indistinguishable from* Hyb_5.

Proof. The difference between the two hybrids is that in Hyb_5, the honest terminal gets its output via the ideal functionality. In more detail, SimHyb instructs the ideal functionality to deliver either the actual output bit y or $\perp$ based on its computation in round 4 of the matching phase. First, note that the strings a, b are picked by SimHyb uniformly at random in round 2, and the adversary only learns $X = (a \cdot \mathsf{IP} + b)$. Observe that in both hybrids, if the adversary sends a string X' in round 3 such that $\mathsf{IP}_{\mathsf{Max}} \leq \frac{(X'-b)}{a}$, the honest terminal outputs $\perp$. The only difference between the two hybrids is if the adversary can send a string $X' \neq X$ in round 3 such that either of the following happens: (i) $\frac{(X'-b)}{a} \leq T$ but $T \leq \frac{(X-b)}{a} \leq \mathsf{IP}_{\mathsf{Max}}$ (OR) (ii) $T \leq \frac{(X'-b)}{a} \leq \mathsf{IP}_{\mathsf{Max}}$ but $\frac{(X-b)}{a} \leq T$. This is because in Hyb_4, the honest terminal's output is based on the value of $\frac{(X'-b)}{a}$ whereas in Hyb_5, it only depends on the value of $\mathsf{IP} = \frac{(X-b)}{a}$. However, since a, b are picked uniformly at random, the probability that the adversary can send a string $X' \neq X$ in round 3 such that either of the two conditions happen is negligible and this completes the proof. □

D.2 Corrupt Provider, Terminal

Consider a semi-honest adversary $\mathcal{A}$ that jointly corrupts the service provider $\mathcal{SP}$ and terminal $\mathcal{T}$. We now describe the simulator Sim. Let $\mathsf{ZK.Sim}, \mathsf{ZK.Sim}_1, \mathsf{ZK.Sim}_2$ denote simulators for the zero knowledge protocols $\Pi_{\mathsf{epk}}, \Pi_{\mathsf{ptxt\text{-}knwldg}}$ and $\Pi_{\mathsf{zk\text{-}L2}}$ respectively.

Enrollment Phase: Sim does the following.

- Pick vector $\overrightarrow{u} = (u_1, \ldots, u_\ell)$ uniformly at random.
- Compute $(\mathsf{esk}, \mathsf{epk}) \leftarrow \mathsf{Plr.Setup}(1^\lambda)$ and send $(\mathsf{epk}, [\![u_1]\!], \ldots, [\![u_\ell]\!])$ to $\mathcal{A}$.
- Run the simulators $\mathsf{ZK.Sim}, \mathsf{ZK.Sim}_1, \mathsf{ZK.Sim}_2$ for the zero knowledge protocols $\Pi_{\mathsf{epk}}, \Pi_{\mathsf{ptxt\text{-}knwldg}}$ and $\Pi_{\mathsf{zk\text{-}L2}}$ from Section 8 where $\mathcal{A}$ is the verifier. For the first proof, the statement is epk and for the next two proofs, the statement is $(\mathsf{epk}, [\![u_1]\!], \ldots, [\![u_\ell]\!])$.
- Receive σ from $\mathcal{A}$.

Matching Phase: Before the phase begins, Sim gets input $(\overrightarrow{w}, \rho, y, L_{\mathsf{term}}(\cdot) = \mathsf{IP} = \langle \overrightarrow{u}, \overrightarrow{w} \rangle)$ where $\overrightarrow{w}$ is $\mathcal{A}$'s input, ρ is $\mathcal{A}$'s randomness, y is the output bit and $L_{\mathsf{term}}(\cdot)$ is the leakage. Sim does the following:

- **Round 1:** Send $(\sigma, \mathsf{epk}, \{[\![u_i^*]\!]\}_{i \in [t]})$ to $\mathcal{A}$.
- **Round 2:** Receive $[\![\widetilde{X}]\!]$ from $\mathcal{A}$.
- **Round 3:**
  (1) Extract strings (a, b) from the adversary's randomness ρ.
  (2) Send $X = (a \cdot \mathsf{IP} + b)$ to $\mathcal{A}$ where $\mathsf{IP} = L_{\mathsf{term}}(\cdot)$.

We show that the above simulation strategy is successful via a series of computationally indistinguishable hybrids where the first hybrid Hyb_0 corresponds to the real world and the last hybrid Hyb_3 corresponds to the ideal world.

(1) Hyb_0**: Real world.** In this hybrid, consider a simulator SimHyb that plays the role of the honest device as in the real world.
(2) Hyb_1**: Simulate Proofs.** In this hybrid, SimHyb runs the simulators $\mathsf{ZK.Sim}, \mathsf{ZK.Sim}_1, \mathsf{ZK.Sim}_2$ for the zero knowledge protocols $\Pi_{\mathsf{epk}}, \Pi_{\mathsf{ptxt\text{-}knwldg}}$ and $\Pi_{\mathsf{zk\text{-}L2}}$ as done by Sim in the enrollment phase of the ideal world.
(3) Hyb_2**: Change computation of** X**.** In this hybrid, in round 3 of the matching phase, SimHyb no longer decrypts $[\![\widetilde{X}]\!]$ to compute X. Instead, it computes $X = (a \cdot \mathsf{IP} + b)$ as done by Sim in the ideal world where (a, b) is extracted from the adversary' randomness ρ and $\mathsf{IP} = L_{\mathsf{term}}(\cdot)$.
(4) Hyb_3**: Switch Ciphertexts.** In this hybrid, SimHyb picks input vector $\overrightarrow{u}$ uniformly at random as done by Sim . This hybrid corresponds to the ideal world.

We now show that every pair of successive hybrids is computationally indistinguishable.

Lemma 11. *Assuming the zero knowledge property for protocols* $\Pi_{epk}, \Pi_{ptxt\text{-}knwldg}$ *and* $\Pi_{zk\text{-}L2}$, Hyb_0 *is computationally indistinguishable from* Hyb_1.

Proof. The only difference between the two hybrids is that in Hyb_0, the proofs of protocols $\Pi_{\mathsf{epk}}, \Pi_{\mathsf{ptxt\text{-}knwldg}}$ and $\Pi_{\mathsf{zk\text{-}L2}}$ are computed by SimHyb by following the honest prover's strategy while in Hyb_1, they are generated using the simulators $\mathsf{ZK.Sim}_1, \mathsf{ZK.Sim}_2$. Thus, if there exists an adversary $\mathcal{A}$ that can distinguish between these two hybrids with non-negligible probability, we can design

a reduction $\mathcal{A}_{\mathsf{zk}}$ that can break the zero knowledge property of either $\Pi_{\mathsf{epk}}, \Pi_{\mathsf{ptxt\text{-}knwldg}}$ or $\Pi_{\mathsf{zk\text{-}L2}}$ which is a contradiction. □

Lemma 12. Hyb_1 *is identical to* Hyb_2.

Proof. In Hyb_2, SimHyb computes and sends $X = (a \cdot \mathsf{IP} + b)$ where (a, b) is extracted from the adversary's randomness ρ and $\mathsf{IP} = L_{\mathsf{term}}(\cdot) = \langle \overrightarrow{u}, \overrightarrow{w} \rangle$. In Hyb_1, SimHyb decrypts $[\![X^*]\!]$ to compute X. Since $\mathcal{A}$ is honest, notice that even in Hyb_1, X is in fact equal to $(a \cdot \mathsf{IP} + b)$. Hence, the two hybrids are identically distributed. □

Lemma 13. *Assuming the semantic security of Paillier encryption,* Hyb_2 *is computationally indistinguishable from* Hyb_3.

Proof. This is identical to the proof of Lemma 5. □

E PROTOCOL 3: PROOF

E.1 Corrupt Device

Consider a malicious adversary $\mathcal{A}$ that corrupts the device $\mathcal{D}$. We now describe the simulator Sim. Let ZK.Ext denote the extractor for the zero knowledge protocol $\Pi_{\mathsf{ptxt\text{-}knwldg}}$. Let Mask.Sim denote the simulator for the packing scheme.

Enrollment Phase. This is identical to the enrollment phase in Section D.1. For completeness, we describe it here. Sim does the following.

- Receive $(\mathsf{epk}, [\![u_1]\!], ..., [\![u_\ell]\!])$ from $\mathcal{A}$.
- Run zero knowledge protocols $\Pi_{\mathsf{epk}}, \Pi_{\mathsf{ptxt\text{-}knwldg}}$ and $\Pi_{\mathsf{zk\text{-}L2}}$ as the verifier with $\mathcal{A}$ as the prover with statement epk for the first proof and statement $(\mathsf{epk}, [\![u_1]\!], ..., [\![u_\ell]\!])$ for the next two proofs. Also, run the extractor ZK.Ext on protocol $\Pi_{\mathsf{ptxt\text{-}knwldg}}$ to extract witness $\{u_i, \rho_i\}_{i=1}^{\ell}$.
- If either of the proofs fail or the extractor ZK.Ext fails, send the message "Failed" to $\mathcal{A}$ and query the ideal functionality $\mathcal{F}$ as part of enrollment with a vector of norm larger than ζ, thereby instructing it to deliver the message "Failed" to the honest service provider.
- Else, do the following:
  (1) Compute $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda)$.
  (2) Compute $([\![u_1^*]\!], \ldots, [\![u_t^*]\!])$ using the algorithm Plr.Add$(\cdot)$ where $u_1^*, \ldots, u_t^*$ is the encoding of $\overrightarrow{u} = (u_1, \ldots, u_\ell)$.
  (3) Send $(\sigma = \mathsf{Sign}(\mathsf{sk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!])))$ to $\mathcal{A}$.
  (4) Query the ideal functionality $\mathcal{F}$ as part of enrollment with vector $\overrightarrow{u}$ extracted from ZK.Ext.

Matching Phase: Sim does the following.

- **Output from ideal functionality $\mathcal{F}$:** Query $\mathcal{F}$ with a new sub-session for the matching phase to receive output bit y.
- **Round 1:** Receive $(\sigma, \{[\![u_i^*]\!]\}_{i \in [t]})$ from $\mathcal{A}$.
  Round 2:
  (1) Output $\perp$ if $\mathsf{Verify}(\mathsf{vk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!]), \sigma) \neq 1$.
  (2) Output "Special Abort" if $\mathsf{Verify}(\mathsf{vk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!]), \sigma) = 1$ (AND) $([\![u_1^*]\!], \ldots, [\![u_t^*]\!], \sigma)$ is not the tuple sent to $\mathcal{A}$ in the Enrollment phase.
  (3) Pick a random value IP satisfying the following: $\mathsf{IP} < T$ if $y = 0$ and $T \leq \mathsf{IP} \leq \mathsf{IP}_{\mathsf{Max}}$ if $y = 1$.
  (4) Pick random strings $a \in \mathbb{Z}_\lambda, b \in \mathbb{Z}_{(\lambda \cdot \log \mathsf{IP}_{\mathsf{Max}} + \lambda)}$, $\mathsf{pad} \in \mathbb{Z}_{\log \mathsf{IP}_{\mathsf{Max}} + \lambda}$.
  (5) Pick $2n$ random strings $X_1, \ldots, X_n, Y_1, \ldots, Y_n$ such that $(X_1 + \ldots + X_n) = (\mathsf{IP} + \mathsf{pad})$ and $(Y_1 + \ldots + Y_n) = (a \cdot \mathsf{IP} + b)$.
  (6) Compute and send $[\![\widetilde{X}]\!] = [\![\mathsf{Mask.Sim}(X_1, \ldots, X_n)]\!]$ and $[\![\widetilde{Y}]\!] = [\![\mathsf{Mask.Sim}(Y_1, \ldots, Y_n)]\!]$ to $\mathcal{A}$.
- **Round 3:** Receive $\mathsf{ot}^{\mathsf{rec}}$ from $\mathcal{A}$.
- **Round 4:** $(\mathcal{T} \rightarrow \mathcal{D})$ $\mathcal{T}$ does the following:
  (1) Compute $(\widetilde{C}, \mathsf{lab}) = \mathsf{Garble.Sim}(y)$.
  (2) Let $(\mathsf{lab}_{\mathsf{pad}}, \mathsf{lab}_a, \mathsf{lab}_b)$ denote the simulated labels for inputs (pad, a, b) to the garbled circuit.
  (3) Let $\mathsf{lab}_{X,Y}$ denote the simulated labels for values (X, Y) that are input by $\mathcal{A}$ to evaluate the circuit. Compute $\mathsf{ot}^{\mathsf{sen}} = \mathsf{OT.Round}_2\mathsf{.Sim}(\mathsf{lab}_{X,Y}, \mathsf{ot}^{\mathsf{rec}})$.
  (4) Send $(\widetilde{C}, \mathsf{lab}_{\mathsf{pad}}, \mathsf{lab}_a, \mathsf{lab}_b, \mathsf{ot}^{\mathsf{sen}})$ to $\mathcal{A}$.
- **Round 5:**
  (1) Receive label $\mathsf{lab}_{\mathsf{y}}$ from $\mathcal{A}$.
  (2) If $\mathsf{lab}_{\mathsf{y}}$ corresponds to the label for output bit y of simulated labels lab, instruct the ideal functionality to deliver output to the honest $\mathcal{T}$. Else, instruct $\mathcal{F}$ to output $\perp$ to the honest terminal.

We now show that the above simulation strategy is successful via a series of computationally indistinguishable hybrids where the first hybrid Hyb_0 corresponds to the real world and the last hybrid Hyb_7 corresponds to the ideal world.

(1) Hyb_0**: Real world.** In this hybrid, consider a simulator SimHyb that plays the role of the honest service provider and terminal as in the real world.
(2) Hyb_1**: Run the Extractor.** In this hybrid, SimHyb runs the extractor ZK.Ext for the zero knowledge protocol $\Pi_{\mathsf{ptxt\text{-}knwldg}}$ as done by Sim in the ideal world to extract witness $\overrightarrow{u} = (u_1, \ldots, u_\ell)$. It also queries the ideal functionality in the enrollment phase as done by Sim.
(3) Hyb_2**: Special Abort for signature forgery.** In this hybrid, in round 2 of the matching phase, SimHyb outputs "Special Abort" if $\mathsf{Verify}(\mathsf{vk}, (\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!]), \sigma) = 1$ (AND) $(\mathsf{epk}, [\![u_1^*]\!], \ldots, [\![u_t^*]\!], \sigma)$ is not the tuple sent to $\mathcal{A}$ in the Enrollment phase, as done by Sim in the ideal world.
(4) Hyb_3**: Simulate OT.** In round 4 of the matching phase, SimHyb computes the OT sender message by using the simulator $\mathsf{OT.Round}_2\mathsf{.Sim}(\cdot)$. That is, $\mathsf{ot}^{\mathsf{sen}} = \mathsf{OT.Round}_2\mathsf{.Sim}(\mathsf{lab}_{X,Y}, \mathsf{ot}^{\mathsf{rec}})$ where $\mathsf{lab}_{X,Y}$ are the labels for the inputs (X, Y) to be used by $\mathcal{A}$ as input to the garbled circuit.
(5) Hyb_4**: Simulate garbled circuit.** In round 4 of the matching phase, SimHyb computes a simulated garbled circuit and simulated labels. That is, $(\widetilde{C}, \mathsf{lab}) = \mathsf{Garble.Sim}(y)$.
(6) Hyb_5**: Switch Ciphertexts.** In this hybrid, SimHyb computes the ciphertexts in the matching phase as follows:
  - Pick a random value IP satisfying the following: $\mathsf{IP} < T$ if $y = 0$ and $T \leq \mathsf{IP} \leq \mathsf{IP}_{\mathsf{Max}}$ if $y = 1$.
  - Pick random strings $a \in \mathbb{Z}_\lambda, b \in \mathbb{Z}_{(\lambda \cdot \log \mathsf{IP}_{\mathsf{Max}} + \lambda)}$, $\mathsf{pad} \in \mathbb{Z}_{\log \mathsf{IP}_{\mathsf{Max}} + \lambda}$.
  - Pick $2n$ random strings $X_1, \ldots, X_n, Y_1, \ldots, Y_n$ such that $(X_1 + \ldots + X_n) = (\mathsf{IP} + \mathsf{pad})$ and $(Y_1 + \ldots + Y_n) = (a \cdot \mathsf{IP} + b)$.

- Compute $[\![\widetilde{z}_0]\!], [\![\widetilde{z}_1]\!]$ using the algorithm Plr.Enc where $\widetilde{z}_0 = \mathsf{Encode}_2(X_1, \ldots, X_n)$ and $\widetilde{z}_1 = \mathsf{Encode}_2(Y_1, \ldots, Y_n)$.
- Pick $2n$ random strings $\rho_1, \ldots, \rho_{2n}$ such that $(\rho_1 + \ldots + \rho_n) = 0$, $(\rho_{n+1} + \ldots + \rho_{2n}) = 0$ and $\forall i, \rho_i \in \mathbb{Z}_{2^k}$. Let $\widetilde{\rho}_0 = \mathsf{Encode}_2(\rho_1, \ldots, \rho_n)$ and $\widetilde{\rho}_1 = \mathsf{Encode}_2(\rho_{n+1}, \ldots, \rho_{2n})$.
- Send $[\![\widetilde{X}]\!] = [\![\mathsf{Mask}(\widetilde{z}_0 + \widetilde{\rho}_0)]\!]$, $[\![\widetilde{Y}]\!] = [\![\mathsf{Mask}(\widetilde{z}_1 + \widetilde{\rho}_1)]\!]$ to $\mathcal{A}$.

(7) Hyb_6**: Masking Simulator.** In this hybrid, SimHyb runs the simulator Mask.Sim of the packing scheme to compute $[\![\widetilde{X}]\!], [\![\widetilde{Y}]\!]$ in round 2 instead of running the honest masking algorithm.

(8) Hyb_7**: Ideal world.** In this hybrid, SimHyb instructs the ideal functionality to either deliver output $\perp$ or the real output to the honest terminal as done by Sim after round 5. This hybrid corresponds to the ideal world.

We now show that every pair of successive hybrids is computationally indistinguishable.

Lemma 14. *Assuming the proof of knowledge property for protocol $\Pi_{ptxt\text{-}knwldg}$ and soundness of protocols $\Pi_{epk}, \Pi_{zk\text{-}L2}$, Hyb_0 is computationally indistinguishable from Hyb_1.*

Proof. This is identical to the proof of Lemma 6. □

Lemma 15. *Assuming the unforgeability of the signature scheme, Hyb_1 is computationally indistinguishable from Hyb_2.*

Proof. This is identical to the proof of Lemma 7. □

Lemma 16. *Assuming the security of the oblivious transfer protocol against a malicious receiver, Hyb_2 is computationally indistinguishable from Hyb_3.*

Proof. The only difference between the two hybrids is the way the OT sender's message in round 4 of the matching phase are generated. In Hyb_2, SimHyb generates the OT sender's message honestly using the labels for the values (X, Y). In Hyb_3, the OT sender's messages are also generated using the simulator $\mathsf{OT.Round}_2.\mathsf{Sim}(\cdot)$. It is easy to see that if there exists an adversary $\mathcal{A}$ that can distinguish between these two hybrids with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{OT}}$ that can break the security of the oblivious transfer protocol against a malicious receiver which is a contradiction. □

Lemma 17. *Assuming the security of the garbling scheme, Hyb_3 is computationally indistinguishable from Hyb_4.*

Proof. The only difference between the two hybrids is the way the garbled circuit and the associated labels are generated. In Hyb_3, SimHyb computes them honestly as follows: $(\widetilde{C}, \mathsf{lab}) = \mathsf{Garble}(C)$ where circuit C is described in Figure 6. In Hyb_4, SimHyb simulates them both as follows: $(\widetilde{C}, \mathsf{lab}) = \mathsf{Garble.Sim}(y)$ where y is the output from the ideal functionality. It is easy to see that if there exists an adversary $\mathcal{A}$ that can distinguish between these two hybrids with non-negligible probability, we can design a reduction $\mathcal{A}_{\widetilde{C}}$ that can break the security of the garbling scheme which is a contradiction. □

Lemma 18. *Assuming the circuit privacy of Paillier encryption, Hyb_4 is computationally indistinguishable from Hyb_5.*

Proof. This is identical to the proof of Lemma 3. □

Lemma 19. *Assuming the security of the packing scheme, Hyb_5 is statistically indistinguishable from Hyb_6.*

Proof. This is identical to the proof of Lemma 4. □

Lemma 20. *Hyb_6 is statistically indistinguishable from Hyb_7.*

Proof. The difference between the two hybrids is that in Hyb_7, the honest terminal gets its output via the ideal functionality. In more detail, SimHyb instructs the ideal functionality to deliver either the actual output bit y or $\perp$ based on its computation after round 5 of the matching phase. Similar to the proof of Lemma 10, the strings a, b, pad are picked by SimHyb uniformly at random in round 2, and the adversary only learns $X = (\mathsf{IP} + \mathsf{pad})$ and $Y = (a \cdot \mathsf{IP} + b)$. Observe that in both hybrids, if the adversary uses input strings (X', Y') to evaluate the garbled circuit such that $(a \cdot (X' - \mathsf{pad}) + b) \neq Y'$, the honest terminal would end up outputting $\perp$. The only difference between the two hybrids is if the adversary can input a pair of strings $(X', Y') \neq (X, Y)$ such that $(a \cdot (X' - \mathsf{pad}) + b) = Y'$ and either of the following happens: (i)$(X' - \mathsf{pad}) \leq T$ but $T \leq (X - \mathsf{pad})$ (or) (ii) $T \leq (X' - \mathsf{pad})$ but $(X - \mathsf{pad}) \leq T$. However, since (a, b, pad) are picked uniformly at random, the probability that this can occur is negligible and this completes the proof. □

E.2 Corrupt Provider, Terminal

Consider a semi-honest adversary $\mathcal{A}$ that jointly corrupts the service provider $\mathcal{SP}$ and terminal $\mathcal{T}$. We now describe the simulator Sim. Let $\mathsf{ZK.Sim}, \mathsf{ZK.Sim}_1, \mathsf{ZK.Sim}_2$ denote the simulator for the zero knowledge protocols $\Pi_{\mathsf{epk}}, \Pi_{\mathsf{ptxt\text{-}knwldg}}$ and $\Pi_{\mathsf{zk\text{-}L2}}$ respectively.

Enrollment Phase: Sim does the following.

- Pick vector $\overrightarrow{u} = (u_1, \ldots, u_\ell)$ uniformly at random.
- Compute $(\mathsf{esk}, \mathsf{epk}) \leftarrow \mathsf{Plr.Setup}(1^\lambda)$ and send $(\mathsf{epk}, [\![u_1]\!], \ldots, [\![u_\ell]\!])$ to $\mathcal{A}$.
- Run the simulators $\mathsf{ZK.Sim}, \mathsf{ZK.Sim}_1, \mathsf{ZK.Sim}_2$ for the zero knowledge protocols $\Pi_{\mathsf{epk}}, \Pi_{\mathsf{ptxt\text{-}knwldg}}$ and $\Pi_{\mathsf{zk\text{-}L2}}$ from Section 8 where $\mathcal{A}$ is the verifier. epk is the statement for the first proof and $(\mathsf{epk}, [\![u_1]\!], \ldots, [\![u_\ell]\!])$ is the statement for the next two.
- Receive σ from $\mathcal{A}$.

Matching Phase: Before the phase begins, Sim gets input $(\overrightarrow{w}, \rho, y)$ where $\overrightarrow{w}$ is $\mathcal{A}$'s input, ρ is $\mathcal{A}$'s randomness and y is the output bit. Sim does the following:

- **Round 1:** Send $(\sigma, \mathsf{epk}, \{[\![u_i^*]\!]\}_{i \in [t]})$ to $\mathcal{A}$.
- **Round 2:** Receive $([\![\widetilde{X}]\!], [\![\widetilde{Y}]\!])$ from $\mathcal{A}$.
- **Round 3:** Send $\mathsf{ot}^{\mathsf{rec}} \leftarrow \mathsf{OT.Round}_1((X, Y); \rho^{\mathsf{ot}})$ to $\mathcal{A}$ where X, Y are picked uniformly at random.
- **Round 4** Receive $(\widetilde{C}, \mathsf{lab}_{\mathsf{pad}}, \mathsf{lab}_a, \mathsf{lab}_b, \mathsf{ot}^{\mathsf{sen}})$ from $\mathcal{A}$.
- **Round 5:** From $\mathcal{A}$'s randomness ρ, extract the label $\mathsf{lab}_{\mathsf{y}}$ of the garbled circuit corresponding to the output bit y. Send $\mathsf{lab}_{\mathsf{y}}$ to $\mathcal{A}$.

We show that the above simulation strategy is successful via a series of computationally indistinguishable hybrids where the

first hybrid Hyb_0 corresponds to the real world and the last hybrid Hyb_4 corresponds to the ideal world.

(1) Hyb_0**: Real world.** In this hybrid, consider a simulator SimHyb that plays the role of the honest device as in the real world.
(2) Hyb_1**: Simulate Proofs.** In this hybrid, SimHyb runs the simulators $\mathsf{ZK.Sim}, \mathsf{ZK.Sim}_1, \mathsf{ZK.Sim}_2$ for the zero knowledge protocols $\Pi_{\mathsf{epk}}, \Pi_{\mathsf{ptxt\text{-}knwldg}}$ and $\Pi_{\mathsf{zk\text{-}L2}}$ as done by Sim in the enrollment phase of the ideal world.
(3) Hyb_2**: Change computation of output label.** In this hybrid, in round 5 of the matching phase, SimHyb no longer evaluates the garbled circuit $\widetilde{C}$ to compute the output label lab_y. Instead, lab_y is extracted from the adversary' randomness ρ corresponding to the output y as done by Sim in the ideal world.
(4) Hyb_3**: Switch OT receiver inputs.** In this hybrid, in round 3 of the matching phase, SimHyb generates the OT receiver's message by picking its inputs X, Y uniformly at random as done by Sim.
(5) Hyb_4**: Switch Ciphertexts.** In this hybrid, SimHyb picks input vector $\overrightarrow{u}$ uniformly at random as done by Sim . This hybrid corresponds to the ideal world.

We now show that every pair of successive hybrids is computationally indistinguishable.

Lemma 21. *Assuming the zero knowledge property for protocols $\Pi_{epk}, \Pi_{ptxt\text{-}knwldg}$ and $\Pi_{zk\text{-}L2}$, Hyb_0 is computationally indistinguishable from Hyb_1.*

Proof. This is identical to the proof of Lemma 11. □

Lemma 22. Hyb_1 *is identical to* Hyb_2.

Proof. In Hyb_2, SimHyb computes the output label lab_y corresponding to output bit y from the randomness used to generate the garbled circuit $\widetilde{C}$. This randomness is extracted from the adversary' randomness ρ. In Hyb_1, SimHyb evaluates the garbled circuit using the labels obtained as output of the OT protocol (with input X, Y) to generate the label lab_y. Since $\mathcal{A}$ is honest, the garbled circuit, OT sender messages in round 4 and the ciphertexts in round 2 are honestly generated and so, notice that even in Hyb_1, lab_y in fact indeed corresponds to the label for the output bit of the protocol - y. Hence, the two hybrids are identically distributed. □

Lemma 23. *Assuming the security of the oblivious transfer protocol against a semi-honest sender,* Hyb_2 *is computationally indistinguishable from* Hyb_3.

Proof. The only difference between the two hybrids is the way the OT receiver's message is generated in round 3 of the matching phase. In Hyb_2, SimHyb generates them using the receiver's inputs (X, Y) computed by decrypting and decoding the ciphertexts received from the sender in round 2 while in Hyb_3, the receiver's inputs (X, Y) are picked uniformly at random. Thus, if there exists an adversary $\mathcal{A}$ that can distinguish between these two hybrids with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{OT}}$ that can break the security of the oblivious transfer protocol against a semi-honest sender which is a contradiction. □

Parameters: A Paillier public key $\mathsf{pk} = (N, g)$, statement $\mathsf{st} = (c_1, \ldots, c_n)$ and witness $\mathsf{wit} = ((x_1, r_1), \ldots, (x_n, r_n))$.
Protocol:

- P samples $s \leftarrow \mathbb{Z}_N$ and sends $[\![s; u]\!] \leftarrow \mathsf{Plr.Enc}(\mathsf{pk}, s; u)$ to V.
- V sends $e_1, \ldots, e_n \leftarrow \mathbb{Z}_N^*$ to P.
- P sends $w := s + \sum_i e_i x_i \bmod N$, and $z := u g^t \prod_i r_i^{e_i} \bmod N^2$ to V where t is defined by $tN + w = s + \sum_i e_i x_i$.
- V outputs accept if $[\![w; z]\!] = [\![s; u]\!] \cdot \prod_i c_i^{e_i}$ and otherwise reject.

Figure 9: ZK protocol $\Pi_{\mathsf{ptxt\text{-}knwldg}}$ for batch proving knowledge of Paillier plaintexts.

Lemma 24. *Assuming the semantic security of Paillier encryption,* Hyb_3 *is computationally indistinguishable from* Hyb_4.

Proof. This is identical to the proof of Lemma 5. □

F ZERO KNOWLEDGE

F.1 Proof: Knowledge of Plaintexts

Honest Verifier Zero Knowledge. Consider a honest-verifier $\mathcal{A}$. The strategy of the simulator Sim on input the verifier's randomness $(e_1, ..., e_n)$ (each of which are randomly chosen in $\mathbb{Z}_N$) is as follows:

- Sample $w \leftarrow \mathbb{Z}_N, z \leftarrow \mathbb{Z}_{N^2}$.
- Define $[\![s; u]\!] := g^w z^N / \prod_i c_i^{e_i}$
- Send $[\![s; u]\!]$ in round 1 and (w, z) in round 3 to $\mathcal{A}$.

Lemma 25. *Assuming the semantic security of Paillier encryption, the above simulation strategy is successful.*

Proof. Observe that the checks performed by $\mathcal{A}$ at the end of the protocol succeed in both the real execution and the simulated one. Therefore, the only difference between the two hybrids is the way the ciphertext $[\![s; u]\!]$ is computed. The rest of the proof is identical to the proof of Lemma 5. □

Proof of knowledge. Consider a malicious prover $\mathcal{A}$. We first describe the strategy of the extractor ZK.Ext below:

- Receive the first message a from Adv.
- Send random $\overrightarrow{e}_0 = (e_{0,1}, \ldots, e_{0,n})$ in round 2 to receive output (w_0, z_0) in round 3 from $\mathcal{A}$.
- For each $i \in [n]$, do the following:
  (1) Rewind the adversary to the beginning of round 2.
  (2) Send $\overrightarrow{e}_i = (e_{0,1}, \ldots, e_{0,i} + 1, e_{0,i+1}, \ldots, e_{0,n})$ in round 2 to receive output (w_i, z_i) in round 3 from $\mathcal{A}$.
  (3) Compute the i^{th} plaintext as $x_i = (w_i - w_0)$ and the corresponding randomness as $r_i = \frac{z_i}{z_0}$.
- Output $\{(x_i, r_i)\}_{i \in [n]}$.

We now argue that the above extractor successfully extracts a witness $\{(x_i, r_i)\}_{i \in [n]}$ with overwhelming probability. We will prove this by reaching a contradiction. Lets assume that there exists a PPT cheating prover $\mathcal{A}$ such that it succeeds in generating accepting proofs even though the extraction of the witness fails. Since the adversary generates accepting proofs, for all $0 \le i \le n$ it must be the case that:

$$g^{w_i} z_i^N = a \cdot \prod_j c_j^{e_i, j}$$

Parameters: A Paillier public key $pk = (N, g)$, a value $y \in \mathbb{Z}$, statement $\text{st} = (c_1 = [\![x_1; r_1]\!], \ldots, c_n = [\![x_n; r_n]\!])$, and witness $\text{wit} = ((x_1, r_1), \ldots, (x_n, r_n))$.

Protocol:

- V uniformly samples a prime $\hat{N} < N$.
- V samples random $\alpha \leftarrow \mathbb{Z}_{2^\lambda}, \rho_1, \ldots, \rho_n, \beta, \hat{\beta} \leftarrow \mathbb{Z}_N$. Then it sends P the following:
  - $[\![w_i]\!] := \alpha[\![x_i]\!] + [\![\rho_i]\!]$ for all $i \in [n]$,
  - $[\![v]\!] := (\sum_{i\in[n]}(-2\alpha\rho_i[\![x_i]\!] - \rho_i^2)) + [\![\beta]\!]$,
  - $[\![\hat{v}]\!] := (\sum_{i\in[n]} c_i[\![x_i]\!]) + d + [\![\hat{\beta}]\!]$ where $c_i := (-2\alpha\rho_i \mod \hat{N})$, $d := (\sum_{i\in[n]} -\rho_i^2 \mod \hat{N})$,
  - $\hat{N}$.
- P decrypts $[\![w_i]\!], [\![v]\!], [\![\hat{v}]\!]$ and sends the following to V:
  - $z := v + \sum_{i\in[n]} w_i^2 \mod N$,
  - $\hat{z} := \hat{v} + \sum_{i\in[n]} w_i^2 \mod \hat{N}$.
- V outputs accept if $z = y\alpha^2 + \beta$ and $\hat{z} \equiv_{\hat{N}} y\alpha^2 + \hat{\beta}$. Otherwise, V outputs reject.

Figure 10: ZK protocol $\Pi_{\text{zk-L2}}$ for proving the L2-norm.

Therefore, for all $1 \le i \le n$:

$$\frac{g^{w_i} z_i^N}{g^{w_0} z_0^N} = \frac{a \cdot \prod_j c_j^{e_i,j}}{a \cdot \prod_j c_j^{e_i,j}}$$

This implies $g^{w_i - w_0}\left(\frac{z_i}{z_0}\right)^N = c_i$ based on the way we pick $\overrightarrow{e}_i$. This implies that c_i is of the form $g^{x_i} r_i^N$ where $x_i = (w_i - w_0)$ and $r_i = \frac{z_i}{z_0}$. This exactly corresponds to a valid Paillier ciphertext and is indeed the tuple output by the extractor ZK.Ext for each index i. Thus, the above extractor successfully extracts a witness $\{(x_i, r_i)\}_{i\in[n]}$ with overwhelming probability and this completes the proof.

F.2 Proof: L2 Norm

Completeness for the first check that $z = y\alpha^2 + \beta$ and Π_{range} is easy to observe. For the second check, the core observation is that the underlying computation never wraps around the Paillier modulus N. Consider the following equalities:

$$\hat{z} = \sum_i \hat{w}_i^2 + \hat{v} \tag{1}$$

$$= \sum_i \alpha^2 x_i^2 + 2\alpha x_i \rho_i + \rho_i^2 - c_i x_i + d + \hat{\beta} \tag{2}$$

$$\equiv_{\hat{N}} \sum_i \alpha^2 x_i^2 + 2\alpha x_i \rho_i + \rho_i^2 - 2\alpha\rho_i x_i - \rho_i^2 + \hat{\beta} \tag{3}$$

$$= \sum_i \alpha^2 x_i^2 + \hat{\beta} \tag{4}$$

Equality 2 holds since $w_i = \alpha x_i + \rho_i$ and $\hat{v} = (\sum_i c_i x_i) + d + \hat{\beta}$ holds over the integers with overwhelming probability. In particular, $\hat{v} \neq (\sum_i c_i x_i) + d + \hat{\beta}$ requires $N - \hat{N} \le \hat{\beta} < N$ which is negligible. Equivalency 3 holds due to c_i, d being defined modulo N'. The final equality holds trivially. Therefore the verifier will always accept for an honest prover.

We now prove honest verifier zero knowledge and soundness.

Honest Verifier Zero Knowledge. Consider a honest-verifier $\mathcal{A}$. The strategy of the simulator Sim is described below. Sim takes as input the statement $[\![x_1]\!], ..., [\![x_n]\!], pk$ and the randomness $(\alpha, \alpha, \beta, \hat{\beta}, \rho_1, ..., \rho_n, \rho_1, ..., \rho_n)$ which the verifier will use.

- Compute and send $(z = y\alpha^2 + \beta, \hat{z} = y\alpha^2 + \hat{\beta})$ to $\mathcal{A}$.

We show that the above simulation strategy is successful.

(1) Hyb_0**: Real world.** In this hybrid, consider a simulator SimHyb that plays the role of the honest prover.
(2) Hyb_1**: Switch** $(z, \hat{z})$**.** In this hybrid, simulator SimHyb computes $z = (y\alpha^2 + \beta)$ and $\hat{z} = (y\alpha^2 + \hat{\beta})$. This now corresponds to the simulated execution.

We now show that every pair of successive hybrids is computationally indistinguishable.

Lemma 26. Hyb_0 *is identical to* Hyb_1.

Proof. In Hyb_1, SimHyb computes and sends $z = (y\alpha^2 + \beta)$ and $\hat{z} = (y\alpha^2 + \hat{\beta})$ where (α, β, bb) are extracted from the adversary's randomness. In Hyb_0, SimHyb decrypts $\{[\![w_i]\!]\}_{i\in[n]}, [\![v]\!]$ to compute $(z, \hat{z})$. Since $\mathcal{A}$ is honest, notice that even in Hyb_0, $z = \hat{z} = (y\alpha^2 + \beta)$ and $\hat{z} = (y\alpha^2 + \hat{\beta})$. Hence, the two hybrids are identically distributed. □

Soundness. Consider a malicious prover $\mathcal{A}$ that produces accepting proof transcripts but $\sum_{i\in[n]} x_i^2 \neq y$. We will show that this event can happen only with negligible probability.

Since the verification succeeds, it implies that:

$$z = y\alpha^2 + \beta \qquad \text{and} \qquad y = \sum_i x_i^2 + e$$

for some $e \in \mathbb{Z}_N^*$. This means that

$$z = \alpha^2 \sum_i x_i^2 + \beta + \alpha^2 e = \gamma + \alpha^2 e \mod N$$

where $\gamma := \alpha^2 \sum_i x_i^2 + \beta$. Note that e is known to the prover. γ is also known to the prover - it decrypts the ciphertexts from the verifier and can compute $\gamma = v + \sum_{i\in[n]} w_i^2$. If the prover can compute $z = \gamma + \alpha^2 e$, this implies that the prover knows α and hence, can also deduce β (since γ is essentially $\alpha^2 \sum_i x_i^2 + \beta$). However, β is a uniformly random value chosen by the verifier and only included in the string v (that is encrypted). Thus, the probability that the prover learns β should be negligible which is a contradiction. Hence, the probability that the prover can compute $z = \gamma + \alpha^2 e$ is negligible which implies that $e = 0$. That is, we can conclude that $z = (\alpha^2 \sum_i x_i^2 + \beta \mod N)$.

Similarly, since the verification succeeds, the second equation $\hat{z} \equiv_{\hat{N}} y\alpha^2 + \hat{\beta}$ also holds. Then, for the same reason as above, it must hold that $\hat{z} = (\alpha^2 \sum_i x_i^2 + \hat{\beta} \mod \hat{N})$.

$$\sum_i x_i^2 \equiv_N \frac{z - \beta}{\alpha^2} \equiv_N y \quad \Longrightarrow \quad \exists t \in \mathbb{Z}^*, \sum_i x_i^2 = y + Nt$$

$$\sum_i x_i^2 \equiv_{N'} \frac{\hat{z} - \hat{\beta}}{\alpha^2} \equiv_{N'} y \quad \Longrightarrow \quad \exists s \in \mathbb{Z}^*, \sum_i x_i^2 = y + N's$$

Rewriting the implication we obtain $Nt = N's$. This means that the prover must choose t to be a multiple of N'. However, since N' is prime and chosen after all of the x_i are fixed, the prover can only do this with negligible probability.