BETA: Biometric Enabled Threshold Authentication

Shashank Agrawal[1], Saikrishna Badrinarayanan[2], Payman Mohassel[3], Pratyay Mukherjee[4], and Sikhar Patranabis[5]

[1]Visa Research, `shashank.agraval@gmail.com`
[2]Visa Research, `bsaikrishna7393@gmail.com`
[3]Facebook, `payman.mohassel@gmail.com`
[4]Visa Research, `pratyay85@gmail.com`
[5]ETH Zürich, `sikharpatranabis@gmail.com`

In the past decades, user authentication has been dominated by server-side password-based solutions that rely on "what users know". This approach is susceptible to breaches and phishing attacks, and poses usability challenges. As a result, the industry is gradually moving to biometric-based client-side solutions that do not store any secret information on servers. This shift necessitates the safe storage of biometric templates and private keys, which are used to generate tokens, on user devices.

We propose a new generic framework called *Biometric Enabled Threshold Authentication* (BETA) to protect sensitive client-side information like biometric templates and cryptographic keys. Towards this, we formally introduce the notion of *Fuzzy Threshold Tokenizer* (FTT) where an initiator can use a "close" biometric measurement to generate an authentication token if at least t (the threshold) devices participate. We require that the devices only talk to the initiator, and not to each other, to capture the way user devices are connected in the real world. We use the universal composability (UC) framework to model the security properties of FTT, including the unforgeability of tokens and the privacy of the biometric values (template and measurement), under a *malicious* adversary. We construct *three* protocols that meet our definition.

Our first two protocols are general feasibility results that work for *any* distance function, *any* threshold t and tolerate the *maximal* (i.e. $t-1$) amount of corruption. They are based on *any* two round UC-secure multi-party computation protocol in the standard model (with a CRS) and threshold fully homomorphic encryption, respectively. We show how to effectively use these primitives to build protocols in a constrained communication model with just four rounds of communication.

For the third protocol, we consider inner-product based distance metrics (cosine similarity, Euclidean distance, etc.) specifically, motivated by the recent interest in its use for face recognition. We use Paillier encryption, efficient NIZKs for specific languages, and a simple garbled circuit to build an efficient protocol for the common case of $n = 3$ devices with one compromised.

Contents

1 Introduction **3**
1.1 Our Contributions . . . 4
1.2 Related Work . . . 7

2 Technical Overview **8**
2.1 MPC based protocol . . . 8
2.2 Threshold FHE based protocol . . . 9
2.3 Cosine Similarity: single corruption . . . 10
2.4 Organization . . . 12

3 Preliminaries **13**
3.1 Threshold Signature . . . 13

4 Formalizing Fuzzy Threshold Tokenizer (FTT) **14**
4.1 Security Definition . . . 14
4.1.1 Ideal Functionality $\mathcal{F}_{\text{FTT}}^{\text{TS}}$ . . . 15
4.2 Discussion . . . 17

5 Any Distance Measure from MPC **18**
5.1 Construction . . . 19

6 Any Distance Measure using Threshold FHE **21**
6.1 Construction . . . 21

7 Cosine Similarity: Single Corruption **23**
7.1 Construction . . . 24
7.2 Euclidean Distance . . . 27

8 Open Problems **27**

A Cryptographic Definitions **32**
A.1 Basic primitives . . . 32
A.2 Additively Homomorphic Encryption . . . 37
A.3 Threshold Fully Homomorphic Encryption . . . 37
A.4 Secure Multiparty Computation . . . 38

B Security Proofs **40**
B.1 Proof of Theorem 1 . . . 40
B.1.1 Description of Simulator . . . 40
B.1.2 Hybrids . . . 42
B.2 Proof of Theorem 3 . . . 46
B.2.1 Description of Simulator . . . 46
B.2.2 Hybrids . . . 50
B.3 Proof of Theorem 5 . . . 63
B.3.1 Description of Simulator . . . 64
B.3.2 Hybrids . . . 66

1 Introduction

Traditionally, password-based authentication has been the dominant approach for authenticating users on the Internet, by relying on "what users know". However, this approach has its fair share of security and usability issues. It typically requires the servers to store a (salted) hash of all passwords, making them susceptible to offline dictionary attacks. Indeed, large-scale password breaches in the wild are extremely common [lis, prc]. Passwords also pose challenging usability problems. High entropy passwords are hard to remember by humans, while low entropy passwords provide little security, and research has shown that introducing complex restrictions on password choices can backfire [GFN+17, Sec A.3].

There are major ongoing efforts in the industry to address some of these issues. For example, "unique" biometric features such as finger-print [pix], facial scans [appa], and iris scans [sam] are increasingly popular first or second factor authentication mechanisms for logging into devices and applications. Studies show that biometrics are much more user-friendly [ess], particularly on mobile devices, as users do not have to remember or enter any secret information. At the same time, a (server-side) breach of biometric data is much more damaging because, unlike passwords, there is no easy way to change biometric information regularly.

Therefore, the industry is shifting away from transmitting or storing user secrets on the server-side. For example, biometric templates and measurements are stored and processed on the client devices where the matching also takes place. A successful match then unlocks a private signing key for a digital signature scheme which is used to generate a token on a fresh challenge. Instead of the user data, the token is transmitted to the server, who only stores a public verification key to verify the tokens. (Throughout the paper, we shall use the terms token and signature interchangeably.) Thus, a server breach does not lead to a loss of sensitive user data.

Most prominently, this is the approach taken by the FIDO Alliance [fid], the world's largest industry-wide effort to enable an interoperable ecosystem of hardware-, mobile- and biometric-based authenticators that can be used by enterprises and service providers. This framework is also widely adopted by major Internet players and incorporated into all major browsers in the form of W3C standard Web Authentication API [w3s].

Hardware-based protection. With biometric data and private keys (for generating tokens) stored on client devices, a primary challenge is to securely protect them. As pointed out before, this is particularly crucial with biometrics since unlike passwords they are not replaceable. The most secure approach for doing so relies on hardware-based solutions such as secure enclaves [appb] that provide physical separation between secrets and applications. However, secure hardware is not available on all devices, can be costly to support at scale, and provides very little programmability.

Software-based protection. Software-based solutions such as white-box cryptography are often based on ad-hoc techniques that are regularly broken [whi]. The provably secure alternative, i.e. cryptographic obfuscation [BGI+01, GGH+13], is not yet practical for real-world use-cases and its mathematical foundation is not yet well understood. A simple alternative approach is to apply "salt-and-hash" techniques, often used to protect passwords, to biometric templates before storing them on the client device. Here, naïve solutions fail because biometric matching is almost always a fuzzy match that checks whether the distance between two vectors is above a threshold or not.

Using fuzzy extractors. It is tempting to think that a better way to implement the hash-and-salt approach for biometric data is through a cryptographic primitive known as *fuzzy extractor* [DRS04, Boy04]. However, as also discussed by Dupont et al. [DHP+18], this approach only works for high-entropy biometric data and is susceptible to offline dictionary attacks.

Distributed cryptography to the rescue. Our work is motivated by the fact that most users own and carry *multiple devices* (laptop, smart-phone, smart-watch, etc.) and have other IoT devices around when authenticating (smart TV, smart-home appliances, etc.). We introduce a new framework for client-side biometric-based authentication that securely distributes both the biometric template as well as the secret signing key among multiple devices. These devices can collectively perform biometric matching and token generation without ever reconstructing the template or the signing key on any one device. We refer to this framework as *Biometric Enabled Threshold Authentication* (BETA for short) and study it at length in this paper.

Before diving deeper into the details, we note that while our primary motivation stems from a client-side authentication mechanism, our framework is quite *generic* and can be used in other settings. For example, it can also be used to protect biometric information on the *server-side* by distributing it among multiple servers who perform the matching and token generation (e.g., for a *single sign-on* authentication token) in a fully distributed manner.

1.1 Our Contributions

To concretely instantiate our framework BETA, we formally introduce the notion of *fuzzy threshold tokenizer* (FTT). We provide a universally composable (UC) security definition for FTT and design several protocols that realize it. We first briefly describe the notion of a Fuzzy Threshold Tokenizer.

Fuzzy Threshold Tokenizer. Consider a set of n parties/devices, a distribution $\mathcal{W}$ over vectors in $\mathbb{Z}_q^\ell$, a threshold t on the number of parties, a distance predicate Dist and an unforgeable threshold signature scheme TS. Initially, in a *global setup* phase, a user generates some public and secret parameters (in a trusted setting), and distributes them amongst the n devices she owns. Further, she also runs the setup of the scheme TS and secret shares the signing key amongst the devices. In an *enrollment* phase, user samples a biometric template $\overrightarrow{\mathbf{w}} \in \mathbb{Z}_q^\ell$ according to $\mathcal{W}$ and securely shares it amongst all the devices. Any set of t devices can, together, completely reconstruct the biometric template $\overrightarrow{\mathbf{w}}$ and the signing key of the threshold signature scheme. Then, during an online *sign on session*, an initiating device P, with a candidate biometric measurement $\overrightarrow{\mathbf{u}}$ as input, can interact in a protocol with a set $\mathcal{S}$ of $(t-1)$ other devices. At the end of this, if $\overrightarrow{\mathbf{u}}$ is "close enough" to the template $\overrightarrow{\mathbf{w}}$ (with respect to distance predicate Dist), the initiating device P obtains a token (signature) on a message of its choice.

It is important to note that we do not allow the other participating $(t-1)$ devices to interact amongst themselves[1] and all communication goes through the initiating device P. This is a critical requirement on the communication model for FTT since in a typical usage scenario, one or two primary devices (e.g., a laptop or a smart-phone) play the role of the initiating device and all other devices are only paired/connected to the primary device. (These

[1]Note that corrupt parties can of course freely interact amongst themselves.

devices may not even be aware of the presence of other devices.) Indeed, this requirement makes the design of constant-round FTT protocols significantly more challenging. Further, in any round of communication, we only allow unidirectional exchange of messages, i.e., either P sends a message to some subset of the other $(t-1)$ devices or vice versa.

Security definition. Consider a probabilistic polynomial time adversary $\mathcal{A}$ that corrupts a set T of devices where $|T| < t$. Informally, the security properties that we wish to capture in an FTT scheme are as follows:

(i) *Privacy of biometric template:* From any sign on session initiated by a corrupt device, $\mathcal{A}$ should not be able to learn any information about the biometric template $\overrightarrow{\mathbf{w}}$ apart from just the output of the predicate $\mathsf{Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}})$ for its choice of measurement $\overrightarrow{\mathbf{u}}$. If the sign on session was initiated by an honest device, $\mathcal{A}$ should learn no information about $\overrightarrow{\mathbf{w}}$. Crucially, we do not impose any restriction on the entropy of the distribution from which the template is picked.

(ii) *Privacy of biometric measurement:* For any sign on session initiated by an honest device, $\mathcal{A}$ should learn no information whatsoever about the measurement $\overrightarrow{\mathbf{u}}$.

(iii) *Token unforgeability*: $\mathcal{A}$ should not be able to compute a valid token (that verifies according to the threshold signature scheme TS) unless it initiated a sign on session on behalf of a corrupt party with a measurement $\overrightarrow{\mathbf{u}}$ such that $\mathsf{Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}) = 1$. Furthermore, $\mathcal{A}$ should only be able to compute *exactly one* token from each such session.

Our *first contribution* is a formal modeling of the security requirements of a fuzzy threshold tokenizer via a real-ideal world security definition in the universal composability (UC) framework [Can01]. We refer the reader to Section 4 for the formal definition and a detailed discussion on its intricacies.

Our next contribution is a design of several protocols that realize this primitive.

Protocol-1(π^{mpc}). Given any threshold signature scheme TS, for any distance measure Dist, any n, t, we construct a four round[2] UC-secure FTT protocol π^{mpc}. Our construction is based on any two-round (over a broadcast channel) UC-secure multi-party computation (MPC) protocol [MW16, PS16, GS18, BL18] in the CRS model that is secure against up to all but one corruption along with other basic primitives. π^{mpc} tolerates up to $(t-1)$ (which is maximal) malicious devices.

Protocol-2 (π^{tfhe}). Given any threshold signature scheme TS, for any distance measure Dist, any n, t, we construct a four round UC-secure FTT protocol π^{tfhe}. Our construction is based on any t out of n threshold fully homomorphic encryption scheme (TFHE) and other basic primitives. Like π^{mpc}, this protocol is secure against $(t-1)$ malicious devices.

[2] Recall that by one communication round, we mean a unidirectional/non-simultaneous message exchange channel over a *peer-to-peer* network. That is, in each round either the initiator sends messages to some subset of the other participating devices or vice versa. In contrast, one round of communication over a *broadcast* channel means that messages are being sent *simultaneously* by multiple (potentially all) parties connected to the channel and all of them receive all the messages sent in that round. All our FTT protocols use peer-to-peer channels which is the default communication model in this paper.

The above two feasibility results are based on two incomparable primitives (two round MPC and threshold FHE). On the one hand, two-round MPC seems like a stronger notion than threshold FHE. But, on the other hand, two-round MPC is known from a variety of assumptions like LWE/DDH/Quadratic Residuosity, while threshold FHE is known only from LWE. Further, the two protocols have very different techniques which may be of independent interest.

Protocol-3 (π^{ip}). We design the third protocol π^{ip} specifically for the cosine similarity distance metric, which has recently been shown to be quite effective for face recognition (CosFace [WWZ+18], SphereFace [LWY+], FaceNet [SKP15]). We pick a threshold of three for this protocol as people nowadays have at least three devices on them most of the time (typically, a laptop, a smart-phone and a smart-watch). π^{ip} is secure in the random oracle model as long as at most one of the devices is compromised. We use Paillier encryption, efficient NIZKs for specific languages, and a simple garbled circuit to build an efficient four-round protocol.

Efficiency analysis of π^{ip}. Finally, we perform a concrete efficiency analysis of our third protocol π^{ip}. We assume that biometric templates and measurements have ℓ features (or elements) and every feature can be represented with m bits. Let λ denote the computational security parameter and s denote the statistical security parameter. In the protocol π^{ip}, we use Paillier encryption scheme to encrypt each feature of the measurement and its product with the shares of the template. The initiator device proves that the ciphertexts are well-formed and the features are of the right length. For Paillier encryption, such proofs can be done efficiently using only $O(\ell m)$ group operations [DJ01, CDN01].

The other devices use the homomorphic properties of Paillier encryption to compute ciphertexts for inner-product shares and some additional values. They are sent back to the initiator but with a MAC on them. Then the other devices generate a garbled circuit that takes the MAC information from them and the decrypted ciphertexts from the initiator to compute if the cosine value exceeds a certain threshold. The garbled circuit constructed here only does 5 multiplications on numbers of length $O(m + \log \ell + s)$. Oblivious transfers can be preprocessed in the setup phase between every pair of parties so that the online phase is quite efficient (only symmetric-key operations). Furthermore, since only one of the two helping devices can be corrupt, only one device needs to transfer the garbled circuit [MRZ15], further reducing the communication overhead. (We have skipped several important details of the protocol here, but they do not affect the complexity analysis. See Section 2.3 for a complete overview of the protocol.)

An alternate design appropach is to use the garbled circuit itself to compute the inner-product. However, there are two disadvantages of this approach. First, it does not scale efficiently with feature vector length. The number of multiplications to be done inside the garbled circuit would be linear in the number of features, or the size of the circuit would be roughly $O(m^2\ell)$. This is an important concern because the number of features in a template can be very large (e.g., see Figure 1 in the NISTIR draft on Ongoing Face Recognition Vendor Test (FRVT) [nis]). Second, the devices would have to prove in zero knowledge that the bits fed as input to the circuit match the secret shares of the template given to them in the enrollment phase. This incurs additional computational overheads.

1.2 Related Work

Fuzzy identity based encryption, introduced by Sahai and Waters [SW05], allows for decrypting a ciphertext encrypted with respect to some identity id if the decryptor possesses the secret key for an identity that almost matches id. However, unlike FTT, at the time of decryption, the decryptor is required to know both identities and which positions match. Recall that one of our main goals is to distribute the biometric template across all devices so that no one device ever learns it.

Function secret sharing, introduced by Boyle et al. [BGI15], enables to share the computation of a function f amongst several users. Another interesting related primitive is homomorphic secret sharing [BGI$^+$18]. However, both these notions don't quite fit in our context because of the limitations on our communication model and the specific security requirements against a malicious adversary.

Secure multiparty computation protocols in the private simultaneous messages model [FKN94, IK97, BIK17a] consider a scenario where there is a client and a set of servers that wish to securely compute a function f on their joint inputs wherein the communication model only involves interaction between the client and each individual server. However, in that model, the adversary can either corrupt the client or a subset of servers but not both.

The work of Dupont et al. [DHP$^+$18] construct a fuzzy password authenticated key exchange protocol where each of the two parties have a password with low entropy. At the end of the protocol, both parties learn the shared secret key only if the two passwords are "close enough" with respect to some distance measure. In our work, we consider the problem of generating signatures and also multiple parties. Another crucial difference is that in their work, both parties hold a copy of the password whereas in our case, the biometric template is distributed between parties and therefore is never exposed to any party. There is also a lot of work on distributed password authenticated key exchange [BCV16] (and the references within) but their setting considers passwords (and so, equality matching) and not biometrics.

There has been a lot of work in developing privacy-preserving ways to compare biometric data [BCP13, BDCG13, DSB17] but it has mostly focused on computing specific distance measures (like Hamming distance) in the two-party setting where each party holds a vector. There has also been some privacy-preserving work in the same communication model as ours [CSS12, JL13, BIK$^+$17b] but it has mainly focused on private aggregation of sensitive user data.

Generating tokens for user authentication in a threshold manner has been of interest lately. Agrawal et al. [AMMR18] propose new threshold encryption schemes and suggest that they can be used for enterprise network authentication, multi-device IoT authentication, etc. Agrawal et al. [AMMM18] and Baum et al. [BFH$^+$19] build threshold protocols for single sign-on authentication. They distribute the password dictionary and the secret key for token generation among a set of servers such that no $t-1$ among them can generate a token on their own. Their setting is somewhat similar to ours but the distance function is just an equality check, which does not work for biometric data. However, they consider both signatures and MACs as tokens.

2 Technical Overview

2.1 MPC based protocol

Emulating General Purpose MPC. Our starting point is the observation that suppose all the parties could freely communicate, then any UC-secure MPC protocol against a malicious adversary in the presence of a broadcast channel would intuitively be very useful in the design of an FTT scheme if we consider the following functionality: the initiator P^* has input $(\mathsf{msg}, S, \overrightarrow{\mathbf{u}})$, every party $P_i \in S$ has input (msg, S), their respective shares of the template $\overrightarrow{\mathbf{w}}$ and the signing key. The functionality outputs a signature on msg to party P^* if $\mathsf{Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}) = 1$ and $|S| = t$. Recently, several works [MW16, PS16, BP16, GS18, BL18] have shown how to construct two round UC-secure MPC protocols in the CRS model in the presence of a broadcast channel from standard cryptographic assumptions. However, the issue with following this intuitive approach is that the communication model of our FTT primitive does not allow all parties to interact amongst each other - in particular, the parties in the set S can't directly talk to each other and all communication has to be routed through the initiator. Armed with this insight, our goal now is to emulate a two round MPC protocol π in our setting.

For simplicity, let us first consider $n = t = 3$. That is, there are three parties: P_1, P_2, P_3. Consider the case when P_1 is the initiator. Now, in the first round of our FTT scheme, P_1 sends msg to both parties. Then, in round 2, we have P_2 and P_3 send their round one messages of the MPC protocol π. In round 3 of our FTT scheme, P_1 sends its own round one message of the MPC protocol to both parties. Along with this, P_1 also sends P_2's round one message to P_3 and vice versa. So now, at the end of round 3 of our FTT scheme, all parties have exchanged their first round messages of protocol π.

Our next observation is that since we care only about P_1 getting output, in the underlying protocol π, only party P_1 needs to receive everyone else's messages in round 2. Therefore, in round 4 of our FTT scheme, P_2 and P_3 can compute their round two messages based on the transcript so far and just send them to P_1. This will enable P_1 to compute the output of protocol π.

Challenges. Unfortunately, the above scheme is insecure. Note that in order to rely on the security of protocol π, we crucially need that for any honest party P_i, every other honest party receives the same first round message on its behalf. Also, we require that all honest parties receive the same messages on behalf of the adversary. In our case, since the communication is being controlled and directed by P_1 instead of a broadcast channel, this need not be true if P_1 was corrupt and P_2, P_3 were honest. Specifically, one of the following two things could occur: (i) P_1 can forward an incorrect version of P_3's round one message of protocol π to P_2 and vice versa. (ii) P_1 could send different copies of its own round 1 message of protocol π to both P_2 and P_3.

Signatures to Solve Challenge 2. To solve the first problem,we simply enforce that P_3 sends a signed copy of its round 1 message of protocol π which is forwarded by P_1 to P_2. Then, P_2 accepts the message to be valid if the signature verifies. In the setup phase, we can distribute a signing key to P_3 and a verification key to everyone, including P_2. Similarly, we can ensure that P_2's actual round 1 message of protocol π was forwarded by P_1 to P_3.

Pseudorandom Functions to Solve Challenge 2. Tackling the second problem is a bit trickier. The idea is instead of enforcing that P_1 send the same round 1 message of protocol

π to both parties, we will instead ensure that P_1 learns their round 2 messages of protocol π only if it did indeed send the same round 1 message of protocol π to both parties. We now describe how to implement this mechanism. Let us denote msg_2 to be P_1's round 1 message of protocol π sent to P_2 and msg_3 (possibly different from msg_2) to be P_1's round 1 message of protocol π sent to P_3. In the setup phase, we distribute two keys k_2, k_3 of a pseudorandom function (PRF) to both P_2, P_3. Now, in round 4 of our FTT scheme, P_3 does the following: instead of sending its round 2 message of protocol π as is, it encrypts this message using a secret key encryption scheme where the key is $\mathsf{PRF}(k_3, \mathsf{msg}_3)$. Then, in round 4, along with its actual message, P_2 also sends $\mathsf{PRF}(k_3, \mathsf{msg}_2)$ which would be the correct key used by P_3 to encrypt its round 2 message of protocol π only if $\mathsf{msg}_2 = \mathsf{msg}_3$. Similarly, we use the key k_2 to ensure that P_2's round 2 message of protocol π is revealed to P_1 only if $\mathsf{msg}_2 = \mathsf{msg}_3$.

The above approach naturally extends for arbitrary n, t. by sharing two PRF keys between every pair of parties. There, each party encrypts its round 2 message of protocol π with a secret key that is an XOR of all the PRF evaluations. There are additional subtle issues when we try to formally prove that the above protocol is UC-secure and we refer the reader to Section 5 for more details.

2.2 Threshold FHE based protocol

The basic idea behind second our protocol is to use an FHE scheme to perform the distance predicate computation between the measurement $\vec{\mathbf{u}}$ and the template $\vec{\mathbf{w}}$. In particular, in the setup phase, we generate the public key pk of an FHE scheme and then in the enrollment phase, each party is given an encryption $\mathsf{ct}_{\vec{\mathbf{w}}}$ of the template. In the sign on phase, an initiator P^* can compute a ciphertext $\mathsf{ct}_{\vec{\mathbf{u}}}$ that encrypts the measurement and send it to all the parties in the set S which will allow them to each individually compute a ciphertext ct^* homomorphically that evaluates $\mathsf{Dist}(\vec{\mathbf{u}}, \vec{\mathbf{w}})$. However, the first challenge is how to decrypt this ciphertext ct^*? In other words, who gets the secret key sk of the FHE scheme in the setup? If sk is given to all parties in S, then they can, of course, decrypt $\mathsf{ct}_{\vec{\mathbf{u}}}$ but that violates privacy of the measurement. On the other hand, if sk is given only to P^*, that allows P^* to decrypt $\mathsf{ct}_{\vec{\mathbf{w}}}$ violating privacy of the template.

Threshold FHE. Observe that this issue can be overcome if somehow the secret key is secret shared amongst all the parties in S in such a way that each of them, using their secret key share sk_i, can produce a partial decryption of ct^* that can then all be combined by P^* to decrypt ct^*. In fact, this is exactly the guarantee of threshold FHE. This brings us to the next issue that if only P^* learns whether $\mathsf{Dist}(\vec{\mathbf{u}}, \vec{\mathbf{w}}) = 1$, how do the parties in S successfully transfer the threshold signature shares? (recall that the transfer should be conditioned upon Dist evaluating to 1) One natural option is, in the homomorphic evaluation of the ciphertext ct, apart from just checking whether $\mathsf{Dist}(\vec{\mathbf{u}}, \vec{\mathbf{w}}) = 1$, perhaps the circuit could then also compute the partial signatures with respect to the threshold signature scheme if the check succeeds. However, the problem then is that, for threshold decryption, there must be a common ciphertext available to each party. In this case, however, each party would generate a partial signature using its own signing key share resulting in a different ciphertext and in turn preventing threshold decryption.

Partial Signatures. To overcome this obstacle, at the beginning of the sign-on phase, each party computes its partial signature σ_i and *information-theoretically* encrypts it via one-time pad with a uniformly sampled one-time key K_i. The parties then transfer the partial signatures in the same round in an encrypted manner without worrying about the result of the decryption. Now, to complete the construction, we develop a mechanism such that:

– Whenever the FHE decryption results in 1, P^* learns the set of one-time secret keys $\{K_i\}$ and hence reconstructs the set of partial signatures $\{\sigma_i\}$.

– Whenever the FHE decryption results in 0, P^* fails to learn any of the one-time secret keys, which in turn ensures that each of the partial signatures remains hidden from P^*.

To achieve that, we do the following: each party additionally broadcasts ct_{K_i}, which is an FHE encryption of its one-time secret key K_i, to every other party during the enrollment phase. Additionally, we use t copies of the FHE circuit being evaluated as follows: the i^{th} circuit outputs K_i if $\mathsf{Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}) = 1$ – that is, this circuit is homomorphically evaluated using the FHE ciphertexts $\mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{K_i}$. [3] Now, at the end of the decryption, if $\mathsf{Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}})$ was indeed equal to 1, P^* learns the set of one-time keys $\{K_i\}$ via homomorphic evaluation and uses these to recover the corresponding partial signatures.

Consider the case where the adversary $\mathcal{A}$ initiates a session with a measurement $\overrightarrow{\mathbf{u}}$ such that $\mathsf{Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}) = 0$. Our security proof formally establishes that the adversary $\mathcal{A}$ learns no information about each one-time key K_i of the honest parties and hence about the corresponding signature share. At a high level, we exploit the simulation and semantic security guarantees of the threshold FHE scheme to: (a) simulate the FHE partial decryptions to correctly output 0 and (b) to switch each ct_{K_i} to be an encryption of 0. At this point, we can switch each K_i to be a uniformly random string and hence "unrecoverable" to $\mathcal{A}$. We refer the reader to Section 6 for more details on this.

NIZKs. One key issue is that parties may not behave honestly - that is, in the first round, P^* might not run the FHE encryption algorithm honestly and similarly, in the second round, each party might not run the FHE partial decryption algorithm honestly which could lead to devastating attacks. To solve this, we require each party to prove honest behavior using a non-interactive zero knowledge argument (NIZK). Finally, as in the previous section, to ensure that P^* sends the same message $\mathsf{ct}_{\overrightarrow{\mathbf{u}}}$ to all parties, we use a signature-based verification strategy, which adds two rounds resulting in a four round protocol.

2.3 Cosine Similarity: single corruption

In this section, we build a protocol for a specific distance measure[4] (Cosine Similarity). It is more efficient compared to our feasibility results. On the flip side, it tolerates only one corruption: that is, our protocol is UC-secure in the Random Oracle model against a malicious adversary that can corrupt only one party. For two vectors $\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}$, $\mathsf{CS.Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}) = \frac{\langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}} \rangle}{||\overrightarrow{\mathbf{u}}|| \cdot ||\overrightarrow{\mathbf{w}}||}$ where $||\overrightarrow{x}||$ denotes the L^2-norm of the vector. $\mathsf{Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}) = 1$ if $\mathsf{CS.Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}) \geq d$ where d is chosen by Dist. Without loss of generality, assume that distribution $\mathcal{W}$ samples vectors $\overrightarrow{\mathbf{w}}$ with $||\overrightarrow{\mathbf{w}}|| = 1$. Then, we check if $\langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}} \rangle > (d \cdot \langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{u}} \rangle)^2$ instead of $\mathsf{CS.Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}) > d$. This syntactic change allows more flexibility.

[3] Note that the creation and broadcasting of these ciphertexts can happen in parallel within a single round of communication between P^* and the other parties in the set S.

[4] Our construction can also be extended to work for the related Euclidean Distance function but we focus on Cosine Similarity in this section.

Distributed Garbling. Our starting point is the following. Suppose we had $t = 2$. Then, we can just directly use Yao's [Yao86] two party semi-honest secure computation protocol as a building block to construct a two round FTT scheme. In the enrollment phase, secret share $\overrightarrow{\mathbf{w}}$ into $\overrightarrow{\mathbf{w}}_1, \overrightarrow{\mathbf{w}}_2$ and give one part to each party. The initiator requests for labels via oblivious transfer (OT) corresponding to his share of $\overrightarrow{\mathbf{w}}$ and input $\overrightarrow{\mathbf{u}}$ while the garbled circuit, which has the other share of $\overrightarrow{\mathbf{w}}$ hardwired, reconstructs $\overrightarrow{\mathbf{w}}$, checks if $\langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}} \rangle > (d \cdot \langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{u}} \rangle)^2$ and if so, outputs a signature. This protocol is secure against a malicious initiator who only has to evaluate the garbled circuit, if we use an OT protocol that is malicious secure in the CRS model. However, to achieve malicious security against the garbler, we would need expensive zero knowledge arguments that prove correctness of the garbled circuit. Now, in order to build an efficient protocol that achieves security against a malicious garbler and to work with threshold $t = 3$, the idea is to distribute the garbling process between two parties.

Consider an initiator P_1 interacting with parties P_2, P_3. We repeat the below process for any initiator and any pair of parties that it must interact with. For ease of exposition, we just consider P_1, P_2, P_3 in this section. Both P_2 and P_3 generate one garbled circuit each using shared randomness generated during setup and the evaluator just checks if the two circuits are identical. Further, both P_2 and P_3 get the share $\overrightarrow{\mathbf{w}}_2$ and a share of the signing key in the enrollment and setup phase respectively. Note that since the adversary can corrupt at most one party, this check would guarantee that the evaluator can learn whether the garbled circuit was honestly generated. In order to ensure that the evaluator does not evaluate both garbled circuits on different inputs, we will also require the garbled circuits to check that P_1's OT receiver queries made to both parties was the same. The above approach is inspired from the three party secure computation protocol of Mohassel et al. [MRZ15].

However, the issue here is that P_1 needs a mechanism to prove in zero knowledge that it is indeed using the share $\overrightarrow{\mathbf{w}}_1$ received in the setup phase as input to the garbled circuit. Moreover, even without this issue, the protocol is computationally quite expensive. For cosine similarity, the garbled circuit will have to perform a lot of expensive operations - for vectors of length ℓ, we would have to perform $O(\ell)$ multiplications inside the garbled circuit. As mentioned in the introduction, because the number of features in a template (ℓ) can be very large for applications like face recognition, our goal is to improve the efficiency and scalability of the above protocol by performing only a constant number of multiplications inside the garbled circuit.

Additive Homomorphic Encryption. Our strategy to build an efficient protocol is to use additional rounds of communication to offload the heavy computation outside the garbled circuit and also along the way, solve the issue of the initiator using the right share $\overrightarrow{\mathbf{w}}_1$. In particular, if we can perform the inner product computation outside the garbled circuit in the first phase of the protocol, then the resulting garbled circuit in the second phase would have to perform only a constant number of operations. In order to do so, we leverage the tool of efficient additively homomorphic encryption schemes [Pai99, ElG84]. In our new protocol, in round 1, the initiator P_1 sends an encryption of $\overrightarrow{\mathbf{u}}$. P_1 can compute $\langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}_1 \rangle$ by itself. Both P_2 and P_3 respond with encryptions of $\langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}_2 \rangle$ computed homomorphically using the same shared randomness. P_1 can decrypt this to compute $\langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}} \rangle$. The parties can then run the garbled circuit based protocol as above in rounds 3 and 4 of our FTT scheme: that is, P_1 requests for labels corresponding to $\langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}} \rangle$ and $\langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{u}} \rangle$ and the garbled circuit does the rest of the check as before. While this protocol is correct and efficient, there are still several issues.

Leaking Inner Product. The first problem is that the inner product $\langle \vec{\mathbf{u}}, \vec{\mathbf{w}} \rangle$ is currently leaked to the initiator P_1 thereby violating the privacy of the template $\vec{\mathbf{w}}$. To prevent this, we need to design a mechanism where no party learns the inner product entirely in the clear and yet the check happens inside the garbled circuit. A natural approach is for P_2 and P_3 to homomorphically compute an encryption of the result $\langle \vec{\mathbf{u}}, \vec{\mathbf{w}}_2 \rangle$ using a very efficient secret key encryption scheme. In our case, just a one time pad suffices. Now, P_1 only learns an encryption of this value and hence the inner product is hidden, while the garbled circuit, with the secret key hardwired into it, can easily decrypt the one-time pad.

Input Consistency. The second major challenge is to ensure that the input on which P_1 wishes to evaluate the garbled circuit is indeed the output of the decryption. If not, P_1 could request to evaluate the garbled circuit on suitably high inputs of his choice, thereby violating unforgeability! In order to prevent this attack, P_2 and P_3 homomorphically compute not just $x = \langle \vec{\mathbf{u}}, \vec{\mathbf{w}}_2 \rangle$ but also a message authentication code (mac) y on the value x using shared randomness generated in the setup phase. We use a simple one time mac that can be computed using linear operations and hence can be done using the additively homomorphic encryption scheme. Now, the garbled circuit also checks that the mac verifies correctly and from the security of the mac, P_1 can not change the input between the two stages. Also, we require P_1 to also send encryptions of $\langle \vec{\mathbf{u}}, \vec{\mathbf{u}} \rangle$ in round 1 so that P_2, P_3 can compute a mac on this as well, thereby preventing P_1 from cheating on this part of the computation too.

Ciphertext Well-formedness. Another important issue to tackle is to ensure that P_1 does indeed send well-formed encryptions. To do so, we rely on efficient zero knowledge arguments from literature [DJ01, CDN01] when instantiating the additively homomorphic encryption scheme with the Paillier encryption scheme [Pai99]. For technical reasons, we also need the homomorphic encryption scheme to be circuit-private. We refer the reader to Section 7 for more details. Observe that in our final protocol, the garbled circuit does only a constant number of multiplications, which makes protocol computationally efficient and scalable.

Optimizations. To further improve the efficiency of our protocol, as done in Mohassel et al. [MRZ15], we will require only one of the two parties P_2, P_3 to actually send the garbled circuit. The other party can just send a hash of the garbled circuit and the initiator can check that the hash values are equal. We refer to Section 7 for more details on this and other optimizations.

2.4 Organization

The rest of the paper is organized as follows. Section 3 presents notations used and preliminary background material. Section 4 formally introduces the notion of FTT and gives a UC security definition for it. Section 5 describes our four round FTT protocol for any distance measure based on UC-secure MPC. Section 6 describes our four round FTT protocol for any distance measure based on threshold FHE. Finally, Section 7 describes our efficient four round FTT protocol for the Euclidean and Cosine Similarity distance measures. The reader may refer Appendix A for additional background material and cryptographic definitions. All proofs of security are deferred to Appendix B. We list some interesting open problems in Section 8.

3 Preliminaries

Let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ denote the n parties and λ the security parameter. Recall that the L^2 norm of a vector $\vec{\mathbf{x}} = (\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_n)$ is defined as $||\vec{\mathbf{x}}|| = \sqrt{\vec{\mathbf{x}}_1^2 + \ldots + \vec{\mathbf{x}}_n^2}$. $\langle \vec{\mathbf{u}}, \vec{\mathbf{w}} \rangle$ denotes the inner product between two vectors $\vec{\mathbf{u}}, \vec{\mathbf{w}}$.

Definition 1. ***(Cosine Similarity)*** *For any two vectors $\vec{\mathbf{u}}, \vec{\mathbf{w}} \in \mathbb{Z}_q^\ell$, the Cosine Similarity between them is defined as follows:*

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

When using this distance measure, we say that $\mathsf{Dist}(\vec{\mathbf{u}}, \vec{\mathbf{w}}) = 1$ *if and only if* $\mathsf{CS.Dist}(\vec{\mathbf{u}}, \vec{\mathbf{w}}) \geq d$ *where d is a parameter specified by* $\mathsf{Dist}(\cdot)$.

3.1 Threshold Signature

Definition 2 (Threshold Signature [Bol03])**.** Let $n, t \in \mathbb{N}$. A threshold signature scheme TS is a tuple of four algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Comb}, \mathsf{Ver})$ that satisfy the correctness condition below.

– $\mathsf{Gen}(1^\lambda, n, t) \to (\mathsf{pp}, \mathsf{vk}, [\![\mathbf{sk}]\!]_n)$. A randomized algorithm that takes n, t and the security parameter λ as input, and generates a verification-key vk and a shared signing-key $[\![\mathsf{sk}]\!]_n$.

– $\mathsf{Sign}(\mathsf{sk}_i, m) =: \sigma_i$. A deterministic algorithm that takes a mesage m and signing key-share sk_i as input and outputs a partial signature σ_i.

– $\mathsf{Comb}(\{\sigma_i\}_{i \in S}) =: \sigma/\perp$. A deterministic algorithm that takes a set of partial signatures $\{\mathsf{sk}_i\}_{i \in S}$ as input and outputs a signature σ or $\perp$ denoting failure.

– $\mathsf{Ver}(\mathsf{vk}, (m, \sigma)) =: 1/0$. 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.

Correctness. For all $\lambda \in \mathbb{N}$, any $t, n \in \mathbb{N}$ such that $t \leq n$, all $(\mathsf{pp}, \mathsf{vk}, [\![\mathbf{sk}]\!]_n)$ generated by $\mathsf{Gen}(1^\lambda, n, t)$, any message m, and any set $S \subseteq [n]$ of size at least t, if $\sigma_i = \mathsf{Sign}(\mathsf{sk}_i, m)$ for $i \in S$, then $\mathsf{Ver}(\mathsf{vk}, (m, \mathsf{Comb}(\{\sigma_i\}_{i \in S}))) = 1$.

Definition 3 (Unforgeability)**.** A threshold signatures scheme $\mathsf{TS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Comb}, \mathsf{Ver})$ is unforgeable if for all $n, t \in \mathbb{N}$, $t \leq n$, and any PPT adversary $\mathcal{A}$, the following game outputs 1 with negligible probability (in security parameter).

– *Initialize.* Run $(\mathsf{pp}, \mathsf{vk}, [\![\mathbf{sk}]\!]_n) \leftarrow \mathsf{Gen}(1^\lambda, n, t)$. Give pp, vk to $\mathcal{A}$. Receive the set of corrupt parties $C \subset [n]$ of size at most $t - 1$ from $\mathcal{A}$. Then give $[\![\mathsf{sk}]\!]_C$ to $\mathcal{A}$. Define $\gamma := t - |C|$. Initiate a list $L := \emptyset$.

– *Signing queries.* On query (m, i) for $i \subseteq [n] \setminus C$ return $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m)$. Run this step as many times $\mathcal{A}$ desires.

– *Building the list.* If the number of signing query of the form (m, i) is at least γ, then insert m into the list L. (This captures that $\mathcal{A}$ has enough information to compute a signature on m.)

– *Output.* Eventually receive output $(m^\star, \sigma^\star)$ from $\mathcal{A}$. Return 1 if and only if $\mathsf{Ver}(\mathsf{vk}, (m^\star, \sigma^\star))$ $= 1$ and $m^\star \notin L$, and 0 otherwise.

4 Formalizing Fuzzy Threshold Tokenizer (FTT)

In this section we formally introduce the notion of *fuzzy threshold tokenizer* (FTT) and give a UC-secure definition. We first describe the algorithms/protocols in the primitive followed by the security definition in the next subsection.

Definition 4 (Fuzzy Threshold Tokenizer (FTT))**.** Given a security parameter $\lambda \in \mathbb{N}$, a threshold signature scheme $\mathsf{TS} = (\mathsf{TS.Gen}, \mathsf{TS.Sign}, \mathsf{TS.Combine}, \mathsf{TS.Verify})$, biometric space parameters $q, \ell \in \mathbb{N}$, a distance predicate $\mathsf{Dist} : \mathbb{Z}_q^\ell \times \mathbb{Z}_q^\ell \to \{0, 1\}$, $n \in \mathbb{N}$ parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and a threshold of parties $t \in [n]$, a FTT scheme/protocol consists of the following tuple $(\mathsf{Setup}, \mathsf{Enrollment}, \mathsf{SignOn}, \mathsf{Ver})$ of algorithms/protocols:

– $\mathsf{Setup}(1^\lambda, n, t, \mathsf{TS}) \to (\mathsf{pp}_{\mathsf{setup}}, \{s_i, \mathsf{sk}_i^{\mathsf{TS}}\}_{i\in[n]}, \mathsf{vk})$: The Setup algorithm is run by a trusted authority. It first runs the key-generation of the threshold signature scheme, $(\{\mathsf{sk}_i^{\mathsf{TS}}\}_{i\in[n]}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda, n, t)$. It generates other public parameters $\mathsf{pp}_{\mathsf{setup}}$ and secret values $s_1, \ldots, s_n$ for each party respectively. It outputs $(\mathsf{vk}, \mathsf{pp}_{\mathsf{setup}})$ to every party and secrets $(\mathsf{sk}_i^{\mathsf{TS}}, s_i)$ to each party $\mathcal{P}_i$. ($\mathsf{pp}_{\mathsf{setup}}$ will be an implicit input in all the algorithms below.)

– $\mathsf{Enrollment}(n, t, q, \ell, \vec{\mathbf{w}}, \mathsf{Dist}) \to (\{a_i\}_{i\in[n]})$: On input the parameters and a template $\vec{\mathbf{w}} \in \mathbb{Z}_q^\ell$ from any party, this algorithm is run by the trusted authority to choose a random sample $\vec{\mathbf{w}} \leftarrow \mathcal{W}$. Then, each party $\mathcal{P}_i$ receives some information a_i.

– $\mathsf{SignOn}(\cdot)$: SignOn is a distributed protocol involving a party P^* along with a set S of parties. Party $\mathcal{P}^*$ has input a measurement $\vec{\mathbf{u}}$, message msg and its secret information $(s_*, \mathsf{sk}_*^{\mathsf{TS}})$. Each party $P_i \in S$ has input $(s_i, \mathsf{sk}_i^{\mathsf{TS}})$. At the end of the protocol, P^* obtains a (private) token Token (or $\perp$, denoting failure) as output. Each party $\mathcal{P}_i \in S$ gets output (msg, i, S). The trusted authority is not involved in this protocol.

– $\mathsf{Ver}(\mathsf{vk}, \mathsf{msg}, \mathsf{Token}) \to \{0, 1\}$: Ver is an algorithm which takes input verification key vk, message m and token Token, runs the verification algorithm of the threshold signature scheme $b := \mathsf{TS.Verify}(\mathsf{vk}, (\mathsf{msg}, \mathsf{Token}))$, and outputs $b \in \{0, 1\}$. This can be run locally by any party or even any external entity.

Communication Model. In the $\mathsf{SignOn}(\cdot)$ protocol, only party $\mathcal{P}^*$ can communicate directly with every party in the set S. We stress that the other parties in S can not interact directly with each other.

4.1 Security Definition

We formally define security via the universal composability (UC) framework [Can01]. Similar to the simplified UC framework [CCL15] we have *a default authenticated channel.* This simplifies the definition of our protocols and can be removed easily by composing with an ideal authenticated channel functionality (e.g. [Can04]).

Consider n parties $P_1, \ldots, P_n$. We consider a *fixed number of parties* in the system throughout the paper. That is, no new party can join the execution subsequently. Let π^{TS} be an FTT scheme parameterized by a threshold signature scheme TS. Consider an adversarial environment $\mathcal{Z}$. We consider a *static corruption* model where there are a fixed set of corrupt parties decided a priori.[5] Informally, it is required that for every adversary $\mathcal{A}$ that corrupts some subset of the parties and participates in the real execution of the protocol, there exist an ideal world adversary Sim, such that for all environments $\mathcal{Z}$, the view of the environment is same in both worlds. We describe it more formally below.

Real world. In the real execution, the FTT protocol π^{TS} is executed in the presence of an adversary $\mathcal{A}$. The adversary $\mathcal{A}$ takes as input the security parameter λ and corrupts a subset of parties. Initially, the Setup algorithm is implemented by a trusted authority. The honest parties follow the instructions of π^{TS}. That is, whenever they receive an "Enrollment" query from $\mathcal{Z}$, they will run the $\mathsf{Enrollment}$ phase of π^{TS}. Similarly, whenever they receive a "Sign on" query from $\mathcal{Z}$ with input $(\mathsf{msg}, \overrightarrow{\mathbf{u}}, S)$, they will initiate a $\mathsf{SignOn}(\cdot)$ protocol with the parties in set S and using input $(\mathsf{msg}, S, \mathsf{sk}_i^{\mathsf{TS}})$. If a $\mathsf{SignOn}(\cdot)$ protocol is initiated with them by any other party, they participate honestly using input $\mathsf{sk}_i^{\mathsf{TS}}$. $\mathcal{A}$ sends all messages of the protocol on behalf of the corrupt parties following any arbitrary polynomial-time strategy.

Ideal world. The ideal world is defined by an trusted ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ described in Figure 1 that interacts with n (say) ideal parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and an ideal world adversary, a.k.a. the simulator Sim via secure (and authenticated) channels. The simulator can corrupt a subset of the parties and may fully control them. The parties (or the simulator) do not interact among themselves. We discuss the ideal functionality in more detail later below.

The environment sets the inputs for all parties including the adversaries in both the worlds. However, the environment does *not* observe any internal interaction. For example, in the ideal world such interactions takes between the ideal functionality and another entity (a party, or the simulator); in real world such interactions take place among the real parties. The output of the environment after an execution consists of the inputs/outputs of the honest parties and the entire view of the adversary (simulator in the ideal world). For ideal functionality $\mathcal{F}$, adversary $\mathcal{A}$, simulator Sim, environment $\mathcal{Z}$ and a protocol π we denote the output of $\mathcal{Z}$ by random variable $\text{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}$in the ideal world and $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ in the real world. We describe the ideal functionality for a FTT scheme in Figure 1 and we elaborate on it in the next subsection.

Definition 5 (UC-Realizing FTT)**.** *Let* TS *be a threshold signature scheme (Definition 3),* $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ *be an ideal functionality as described in Figure 1 and* π^{TS} *be a FTT scheme.* π^{TS} *UC-realizes* $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ *if* for any *real world PPT adversary* $\mathcal{A}$*,* there exists *a PPT simulator* Sim *such that* for all *environments* $\mathcal{Z}$*,*

$$\text{IDEAL}_{\mathcal{F}_{\text{FTT}}^{\mathsf{TS}},\mathsf{Sim},\mathcal{Z}} \approx_c \text{REAL}_{\pi^{\mathsf{TS}},\mathcal{A},\mathcal{Z}}$$

Intuitively, for any adversary there should be a simulator that can simulate its behavior such that no environment can distinguish between these two worlds. Also, our definition can also capture setup assumptions such as random oracles by considering a $\mathcal{G}$-hybrid model with an ideal functionality $\mathcal{G}$ for the setup.

4.1.1 Ideal Functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$

The ideal functionality we consider is presented formally in Figure 1. We provide an informal exposition here. Contrary to most of the UC ideal functionalities, our ideal functionality

[5] However, we allow the attacker to decide on the corrupt set adaptively after receiving the public values.

Ideal Functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$

Given a threshold signature scheme $(\mathsf{TS.Gen}, \mathsf{TS.Sign}, \mathsf{TS.Combine}, \mathsf{TS.Verify})$, the functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ is parameterized by a security parameter $\lambda \in \mathbb{N}$, biometric space parameters $q, \ell \in \mathbb{N}$, a distance predicate $\mathsf{Dist} : \mathbb{Z}_q^\ell \times \mathbb{Z}_q^\ell \to \{0,1\}$, number of parties $n \in \mathbb{N}$ and a threshold of parties $t \in [n]$. It interacts with an ideal adversary (the simulator) Sim and n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ via the following queries.

- **On receiving a query of the form (“`Setup`”, sid, aux) from** Sim, do as follows only if sid is unmarked:
  1. run $(\mathsf{vk}, \{\mathsf{sk}_i^{\mathsf{TS}}\}_{i\in[n]}) \leftarrow \mathsf{TS.Gen}(1^\lambda)$;
  2. send (“`VerKey`”$, \mathsf{sid}, \mathsf{vk}, \mathsf{aux}$) to all parties including Sim;
  3. receive a response (“`Corrupt`”$, \mathsf{sid}, C \subseteq [n]$) from Sim;
  4. send (“`KeyShares`”$, \mathsf{sid}, \{\mathsf{sk}_i^{\mathsf{TS}}\}_{i\in C}$) to Sim;
  5. store the tuple $(\mathsf{sid}, \mathsf{vk}, \{\mathsf{sk}_i^{\mathsf{TS}}\}_{i\in[n]})$ and mark this session as “Live”.

- **On receiving a query of the form (“`Enroll`”, sid, $\vec{\mathbf{w}}$) from** $\mathcal{P}$, only if sid is marked “Live”:
  1. store the tuple $(\mathsf{sid}, \vec{\mathbf{w}})$;
  2. send (“`Enrolled`”$, \mathsf{sid}$) to everyone and mark sid as “Enrolled”.

- **On receiving a query of the form (“`SignOn`”, sid, vk, m, $\mathcal{P}$, $\vec{\mathbf{u}}$, $S \subseteq [n]$) from** $\mathcal{P}$, if the session sid is not marked “Enrolled”, ignore this query. Else, retrieve the record $(\mathsf{sid}, \mathsf{pp}, \mathsf{vk}, \{\mathsf{sk}_i^{\mathsf{TS}}\}_{i\in[n]})$ and let $\{\mathcal{P}_j\}_{j\in S}$ be the parties in the set. Send (“`Signing Req`”$, \mathsf{sid}, m, \mathcal{P}, S$) to all $\{\mathcal{P}_j\}_{j\in S}$ in the set S. Then:
  1. **if** $\mathsf{Dist}(\vec{\mathbf{u}}, \vec{\mathbf{w}}) = 1$, $|S| = t$ and all parties in the set S send (“`Agreed`”$, \mathsf{sid}, m, \mathcal{P}$) then: generate $\{\mathsf{Token}_j \leftarrow \mathsf{TS.Sign}(\mathsf{sk}_j^{\mathsf{TS}}, \mathsf{msg})\}_{j\in S}$ and send (“`Parts`”$, \mathsf{sid}, \{\mathsf{Token}_j\}_{j\in S}$) to $\mathcal{P}$.
  2. **otherwise**, return (“`SignOn failed`”$, \mathsf{sid}, \vec{\mathbf{u}}$) to $\mathcal{P}$.

Figure 1: The ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$.

$\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ is parameterized with a threshold signature scheme $\mathsf{TS} = (\mathsf{TS.Gen}, \mathsf{TS.Sign}, \mathsf{TS.Combine}, \mathsf{TS.Verify})$ (see discussion about this choice later in this section). The ideal functionality is parameterized with a distance predicate Dist, which takes two vectors, a template and a candidate measurement and returns 1 if and only if the two vectors are “close”. Additionally, the functionality is parameterized with other standard parameters and a probability distribution over the biometric vectors.

The ideal functionality has an interface to handle queries from different parties. For a particular session, the first query it responds to “`Setup`” from Sim. In response, the functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ generates the key pairs of the given threshold signature scheme, gives the shares for the corrupt parties to the simulator and marks this session “Live”. Then, an “`Enroll`” query can be made by any party. $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ chooses a template $\vec{\mathbf{w}}$ at random from the distribution $\mathcal{W}$, stores it and marks the session as “Enrolled”.

For any “Enrolled” session, $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ can receive many “`SignOn`” queries (the previous two queries are allowed only once per session). This is ensured by not marking the session in response to any such query. The “`SignOn`” query from a party $\mathcal{P}_i$ contains a set S of parties (i.e. their identities), a message to be signed and a candidate measurement $\vec{\mathbf{u}}$. In response, $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ reaches out to all parties in S for a response. Then, $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ checks whether the measurement $\vec{\mathbf{u}}$ is "close enough" by computing $b := \mathsf{Dist}(\vec{\mathbf{u}}, \vec{\mathbf{w}})$. If b is 1, the size of the set $S = t$ and all parties in S send an agreement response, $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ generates the partial signatures (tokens) on behalf of the parties in S and sends them *only to the initiator* $\mathcal{P}_i$; otherwise, it sends a message denoting failure to $\mathcal{P}_i$. Note that the signatures (or even the failure messages) are not sent to the simulator unless the initiator $\mathcal{P}_i$ is corrupt. This is crucial for our definition (see more discussion in Section 4.2.) as it ensures that if a "`SignOn`" query is initiated by an honest party, then the simulator does not obtain *anything* directly, except when there is a corrupt party in S via which it knows such a query has been made and only learns the tuple $(m, \mathcal{P}_i, S)$ corresponding to the query. In fact, no one except the initiator learns whether "`SignOn`" was successful. Intuitively, a protocol realizing $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ must guarantee that a corrupt party can *not* compute a valid sign-on token (signature) just by participating in a session started by an honest party. In our definition of $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$, such a token would be considered as a forgery. To the best of our knowledge, this feature has not been considered in prior works on threshold signatures.

4.2 Discussion

Use of Threshold Signatures. Note that our definition and the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ are both parameterized by a threshold signature scheme. A tuple of algorithms/protocols is said to be a UC-secure FTT scheme only when they securely realize the ideal functionality while both are instantiated with the same threshold signature scheme. We emphasize that we do not rely on the security/unforgeability of the threshold signature in the UC definition. A protocol can be secure according to the definition even if the underlying threshold signature scheme is insecure. However, the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ possesses the desired properties (intuitively explained in the introduction) only if instantiated with a correct and secure threshold signature. Thus, to achieve a meaningful notion of correctness and security from our FTT primitive, we require that the underlying threshold signature scheme is correct and secure.

Privacy of Tokens/Signatures. A main feature of our framework is privacy of tokens (signatures), in that only the party that initiates a sign-on session receives a token. Other participants in that session learn nothing about the token, not even whether the session is successful in generating one. As discussed in the introduction, it is crucial for our application that no corrupt party be able to compute a valid token just by participating in a sign-on session initiated by an honest party (who possibly has a close biometric match). To capture this in the ideal functionality we must enforce that such a token be counted as a forgery. We do so by first explicitly allowing *only* the initiator of a "`SignOn`" query to obtain a response (consisting of partial signatures) from the functionality. Further, contrary to the UC-definition of signatures, we do *not* allow the simulator to generate signatures. Otherwise, in that particular scenario (honest initiator and corrupt participants), simulator on learning the message, can just compute a token on its own. Instead, since in $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$, the functionality generates all signatures, the simulator would be unable to obtain such signatures. The only way for the simulator to obtain a valid token is to make an explicit "`SignOn`" query via a corrupt party (with a "close" measurement) which exactly captures the desired guarantee. To formalize this, we parameterize $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ with a concrete threshold signature, on which the real world (Definition 4) protocol too must be based on.

Comparison with Fuzzy PAKE [DHP+18]. We emphasize that our setting is fundamentally different from the exciting notion of Fuzzy Password Authenticated Key Exchange (PAKE) recently introduced by Dupont et al. [DHP+18]. In their two party primitive, the goal is to establish a shared secret key to be used for subsequent mutual authentication (as in traditional key exchange). A common (random) key is established only when both parties use "close" enough biometric data. Obviously, authenticated channels can not be assumed in their setting and they mainly try to protect against a man-in-the-middle adversary. Further, they require that the protocol must detect (and mark the session as interrupted) if a "bad" (not close) biometric input is used. To do so, they also capture offline attacks on low-entropy biometric distributions explicitly through a separate "`TestPassword`" query (also present in an earlier work [CHK+05]) from the simulator.

Our functionality has an in-built authenticated channel between every pair of parties as the goal is to generate a token in a distributed manner that may be used for authenticating to an external server. In our case, offline attacks are already implicitly captured. That is, since the template stays inside the ideal functionality, the simulator's best bet to get a valid token is to correctly guess a biometric measurement and use a "`SignOn`" query. The success probability increases with each such query. For low entropy biometrics, the simulator may learn a close match after only a few queries. However, it is crucial to note that our primitive guarantees that even after potentially learning the biometric template, any party can generate a valid token only via an online sign-on query (if, of course, a secure threshold signature is used). Also, we do not require that sessions are to be interrupted if a "bad" biometric input is used - to the contrary, no party in our system is allowed to learn whether the querying party was successful in the sign on session.

Comparison with UC-secure blind signatures [KZ08]. To capture blind signatures, it is necessary to hide the message (to be signed) from the simulator. On the other hand, while they also use a concrete signature scheme within the ideal functionality, the simulator gets to pick the particular signature scheme and choose the signing key. In contrast, in our setting, we want the simulator to learn the message to be signed, but not to choose the signature scheme or to be able to generate signatures (in particular when the message is chosen by an honest party).

5 Any Distance Measure from MPC

In this section, we show how to construct a four round secure fuzzy threshold tokenizer using any two round malicious UC-secure MPC protocol in a broadcast channel as the main technical tool. Our tokenizer scheme satisfies Definition 1 for any n, t, for any distance measure. Formally, we show the following theorem:

Theorem 1. *Assuming unforgeable threshold signatures and a two round UC-secure MPC protocols in the CRS model in a broadcast channel, there exists a four round secure fuzzy threshold tokenizer protocol for any n, t and any distance predicate.*

Such two round MPC protocols can be built assuming DDH/LWE/QR/N^{th} Residuosity [MW16, PS16, GS18, BL18]. Threshold signatures can be built assuming LWE/Gap-DDH/RSA [BGG+18, Bol03, Sho00]. Instantiating this, we get the following corollary:

Corollary 2. *Assuming LWE, there exists a four round secure FTT protocol for any n, t and any distance predicate.*

We describe the construction below and defer the proof to Appendix B.1.

5.1 Construction

Notation. Let π be a two round UC-secure MPC protocol in the CRS model in the presence of a broadcast channel that is secure against a malicious adversary that can corrupt upto $(t-1)$ parties. Let $\pi.\mathsf{Setup}$ denote the algorithm used to generate the CRS. Let $(\pi.\mathsf{Round}_1, \pi.\mathsf{Round}_2)$ denote the algorithms used by any party to compute the messages in each of the two rounds and $\pi.\mathsf{Out}$ denote the algorithm to compute the final output. Let $(\mathsf{TS.Gen}, \mathsf{TS.Sign}, \mathsf{TS.Combine}, \mathsf{TS.Verify})$ be a threshold signature scheme, $(\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ be a secret key encryption scheme, $(\mathsf{Share}, \mathsf{Recon})$ be a (t,n) threshold secret sharing scheme and PRF be a pseudorandom function. We now describe the construction of our four round secure fuzzy threshold tokenizer protocol π^{Any} for any n and t.

Setup: The following algorithm is executed by a trusted authority:

– Generate $\mathsf{crs} \leftarrow \pi.\mathsf{Setup}(1^\lambda)$.

– For each $i \in [n]$, compute $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$.

– For every $i, j \in [n]$, compute $(\mathsf{k}^{\mathsf{PRF}}_{i,j}, \mathsf{k}^{\mathsf{PRF}}_{j,i})$ as uniformly random strings.

– Compute $(\mathsf{pp}^{\mathsf{TS}}, \mathsf{vk}^{\mathsf{TS}}, \mathsf{sk}^{\mathsf{TS}}_1, \ldots, \mathsf{sk}^{\mathsf{TS}}_n) \leftarrow \mathsf{TS.Gen}(1^\lambda, n, t)$.

– For each $i \in [n]$, give $(\mathsf{crs}, \mathsf{pp}^{\mathsf{TS}}, \mathsf{vk}^{\mathsf{TS}}, \mathsf{sk}^{\mathsf{TS}}_i, \mathsf{sk}_i, \{\mathsf{vk}_j\}_{j\in[n]}, \{\mathsf{k}^{\mathsf{PRF}}_{j,i}, \mathsf{k}^{\mathsf{PRF}}_{i,j}\}_{j\in[n]})$ to party $\mathcal{P}_i$.

Enrollment: In this phase, any party $\mathcal{P}_i$ that wishes to enroll queries the trusted authority which then does the following:

– Sample a random vector $\overrightarrow{\mathbf{w}}$ from the distribution $\mathcal{W}$.

– Compute $(\overrightarrow{\mathbf{w}}_1, \ldots, \overrightarrow{\mathbf{w}}_n) \leftarrow \mathsf{Share}(1^\lambda, \overrightarrow{\mathbf{w}}, n, t)$.

– For each $i \in [n]$, give $(\overrightarrow{\mathbf{w}}_i)$ to party $\mathcal{P}_i$.

SignOn Phase: In the SignOn phase, let's consider party $\mathcal{P}^*$ that uses input vector $\overrightarrow{\mathbf{u}}$, a message msg on which it wants a token. $\mathcal{P}^*$ interacts with the other parties in the below four round protocol.

Round 1: $(\mathcal{P}^* \rightarrow)$ [6] Party $\mathcal{P}^*$ does the following:

1. Pick a set S consisting of t parties amongst $\mathcal{P}_1, \ldots, \mathcal{P}_n$. For simplicity, without loss of generality, we assume that $\mathcal{P}^*$ is also part of set S.

2. To each party $\mathcal{P}_i \in S$, send (msg, S).

Round 2: $(\rightarrow \mathcal{P}^*)$ Each Party $\mathcal{P}_i \in S$ (except $\mathcal{P}^*$) does the following:

1. Participate in an execution of protocol π with parties in set S using input $\mathsf{y}_i = (\overrightarrow{\mathbf{w}}_i, \mathsf{sk}^{\mathsf{TS}}_i)$ and randomness r_i to compute circuit $\mathcal{C}$ defined in Figure 2. Compute first round message $\mathsf{msg}_{1,i} \leftarrow \pi.\mathsf{Round}_1(\mathsf{y}_i; \mathsf{r}_i)$.

2. Compute $\sigma_{1,i} = \mathsf{Sign}(\mathsf{sk}_i, \mathsf{msg}_{1,i})$.

[6]The arrowhead denotes that in this round messages are outgoing from party $\mathcal{P}^*$.

3. Send $(\mathsf{msg}_{1,i}, \sigma_{1,i})$ to party $\mathcal{P}^*$.

Round 3: $(\mathcal{P}^* \rightarrow)$ Party $\mathcal{P}^*$ does the following:

1. Let $\mathsf{Trans}_{\mathsf{DiFuz}}$ denote the set of messages received in round 2.
2. Participate in an execution of protocol π with parties in set S using input $\mathsf{y}_* = (\overrightarrow{\mathbf{w}}_*, \mathsf{sk}_*^{\mathsf{TS}}, \overrightarrow{\mathbf{u}}, \mathsf{msg})$ and randomness r_* to compute circuit $\mathcal{C}$ defined in Figure 2. Compute first round message $\mathsf{msg}_{1,*} \leftarrow \pi.\mathsf{Round}_1(\mathsf{y}_*; \mathsf{r}_*)$.
3. To each party $\mathcal{P}_i \in S$, send $(\mathsf{Trans}_{\mathsf{DiFuz}}, \mathsf{msg}_{1,*})$.

Round 4: $(\rightarrow \mathcal{P}^*)$ Each Party $\mathcal{P}_i \in S$ (except $\mathcal{P}^*$) does the following:

1. Let $\mathsf{Trans}_{\mathsf{DiFuz}}$ consist of a set of messages of the form $(\mathsf{msg}_{1,j}, \sigma_{1,j})$, $\forall j \in S \setminus \mathcal{P}^*$. Output $\perp$ if $\mathsf{Verify}(\mathsf{vk}_j, \mathsf{msg}_{1,j}, \sigma_{1,j}) \neq 1$.
2. Let $\tau_1 = \{\mathsf{msg}_{1,j}\}_{j \in S}$ denote the transcript of protocol π after round 1. Compute second round message $\mathsf{msg}_{2,i} \leftarrow \pi.\mathsf{Round}_2(\mathsf{y}_i, \tau_1; \mathsf{r}_i)$.
3. Let $(\mathsf{Trans}_{\mathsf{DiFuz}}, \mathsf{msg}_{1,*})$ denote the message received from $\mathcal{P}^*$ in round 3. Compute $\mathsf{ek}_i = \oplus_{j \in S} \mathsf{PRF}(\mathsf{k}_{i,j}^{\mathsf{PRF}}, \mathsf{msg}_{1,*})$ and $\mathsf{ct}_i = \mathsf{SKE.Enc}(\mathsf{ek}_i, \mathsf{msg}_{2,i})$.
4. For each party $\mathcal{P}_j \in S$, compute $\mathsf{ek}_{j,i} = \mathsf{PRF}(\mathsf{k}_{j,i}^{\mathsf{PRF}}, \mathsf{msg}_{1,*})$.
5. Send $(\mathsf{ct}_i, \{\mathsf{ek}_{j,i}\}_{j \in S})$ to $\mathcal{P}^*$.

Output Computation: Every party $\mathcal{P}_j \in S$ outputs $(\mathsf{msg}, \mathcal{P}^*, S)$. Additionally, party $\mathcal{P}^*$ does the following to generate a token:

1. For each party $\mathcal{P}_j \in S$, compute $\mathsf{ek}_j = \oplus_{j \in S} \mathsf{ek}_{j,i}$, $\mathsf{msg}_{2,j} = \mathsf{SKE.Dec}(\mathsf{ek}_j, \mathsf{ct}_j)$.
2. Let τ_2 denote the transcript of protocol π after round 2. Compute the output of π: $\{\mathsf{Token}_i\}_{i \in S} \leftarrow \pi.\mathsf{Out}(\mathsf{y}_*, \tau_2; \mathsf{r}_*)$.
3. Reconstruct the signature as $\mathsf{Token} = \mathsf{TS.Combine}(\{\mathsf{Token}_i\}_{i \in S})$.
4. If $\mathsf{TS.Verify}(\mathsf{vk}^{\mathsf{TS}}, \mathsf{msg}, \mathsf{Token}) = 1$, then output $\{\mathsf{Token}_i\}_{i \in S}$. Else, output $\perp$.

Inputs:

- Party $\mathcal{P}_i \in S$ has input $(\overrightarrow{\mathbf{w}}_i, \mathsf{sk}_i^{\mathsf{TS}})$.
- Party $\mathcal{P}^* \in S$ additionally has input $(\overrightarrow{\mathbf{u}}, \mathsf{msg})$.

Computation:

- Compute $\overrightarrow{\mathbf{w}} = \mathsf{Recon}(\{\overrightarrow{\mathbf{w}}_i\}_{i \in S})$. Output $\perp$ to $\mathcal{P}^*$ if the reconstruction fails or if $\mathsf{Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}) = 0$.
- Compute $\mathsf{Token}_i = \mathsf{TS.Sign}(\mathsf{sk}_i^{\mathsf{TS}}, \mathsf{msg})$. Output $\{\mathsf{Token}_i\}_{i \in S}$ to party $\mathcal{P}^*$.

Figure 2: Circuit $\mathcal{C}$

Token Verification: Given a verification key $\mathsf{vk}^{\mathsf{TS}}$, message msg and a token $\{\mathsf{Token}_i\}_{i \in S}$, where $|S| = t$, the token verification algorithm does the following:

1. Compute $\mathsf{Token} \leftarrow \mathsf{TS.Combine}(\{\mathsf{Token}_i\}_{i \in S})$.
2. Output 1 if $\mathsf{TS.Verify}(\mathsf{vk}^{\mathsf{TS}}, \mathsf{msg}, \mathsf{Token}) = 1$. Else, output 0.

6 Any Distance Measure using Threshold FHE

In this section, we construct a FTT protocol for any distance measure using any fully homomorphic encryption (FHE) scheme with threshold decryption. Our token generation protocol satisfies the definition in Section 4 for any n, t, and works for any distance measure. Formally, we show the following theorem:

Theorem 3. *Assuming threshold fully-homomorphic encryption, non-interactive zero knowledge argument of knowledge (NIZK) and unforgeable threshold signatures, there exists a four round secure FTT protocol for any n, t and any distance predicate.*

Threshold FHE, NIZKs and unforgeable threshold signatures can be built assuming LWE [BGG+18, PS19]. Instantiating this, we get the following corollary:

Corollary 4. *Assuming LWE, there exists a four round secure FTT protocol for any n, t and any distance predicate.*

We describe the construction below and defer the proof to Appendix B.2.

6.1 Construction

Notation. Let ($\mathsf{TFHE.Gen}$, $\mathsf{TFHE.Enc}$, $\mathsf{TFHE.PartialDec}$, $\mathsf{TFHE.Eval}$, $\mathsf{TFHE.Combine}$) be a threshold FHE scheme and let $(\mathsf{TS.Gen}, \mathsf{TS.Sign}, \mathsf{TS.Combine}, \mathsf{TS.Verify})$ be a threshold signature scheme. Let $(\mathsf{Prove}, \mathsf{Verify})$ be a NIZK scheme and $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a strongly-unforgeable digital signature scheme and Commit be a non-interactive commitment scheme. We now describe the construction of our four round secure FTT protocol $\pi^{\mathsf{Any-TFHE}}$ for any n and k.

Setup Phase: The following algorithm is executed by a trusted authority:

- Generate $(\mathsf{pk}^{\mathsf{TFHE}}, \mathsf{sk}_1^{\mathsf{TFHE}}, \ldots, \mathsf{sk}_N^{\mathsf{TFHE}}) \leftarrow \mathsf{TFHE.Gen}(1^\lambda, n, t)$ and $(\mathsf{pp}^{\mathsf{TS}}, \mathsf{vk}^{\mathsf{TS}}, \mathsf{sk}_1^{\mathsf{TS}}, \ldots, \mathsf{sk}_n^{\mathsf{TS}}) \leftarrow \mathsf{TS.Gen}(1^\lambda, n, t)$.
- For each $i \in [n]$, compute $\mathsf{com}_i \leftarrow \mathsf{Commit}(\mathsf{sk}_i^{\mathsf{TFHE}}; \mathsf{r}_i^{\mathsf{com}})$ and $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$.
- For each $i \in [n]$, give the following to party $\mathcal{P}_i$: $(\mathsf{pk}^{\mathsf{TFHE}}, \mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{pp}^{\mathsf{TS}}, \mathsf{vk}^{\mathsf{TS}}, \mathsf{sk}_i^{\mathsf{TS}}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_n), \mathsf{sk}_i, (\mathsf{com}_1, \ldots, \mathsf{com}_n), \mathsf{r}_i^{\mathsf{com}})$.

Enrollment: In this phase, any party $\mathcal{P}_i$ that wishes to register a fresh template queries the trusted authority, which then executes the following algorithm:

- Sample a template $\vec{\mathbf{w}}$ from the distribution $\mathcal{W}$ over $\{0,1\}^\ell$.
- Compute and give $\mathsf{ct}_{\vec{\mathbf{w}}}$ to each party $\mathcal{P}_i$, where $\mathsf{ct}_{\vec{\mathbf{w}}} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \vec{\mathbf{w}})$.

SignOn Phase: In the SignOn phase, let's consider party $\mathcal{P}^*$ that uses input vector $\vec{\mathbf{u}} \in \{0,1\}^\ell$ and a message msg on which it wants a token. $\mathcal{P}^*$ interacts with the other parties in the below four round protocol.

- **Round 1:** $(\mathcal{P}^* \rightarrow)$ [7] Party $\mathcal{P}^*$ does the following:
  1. Compute ciphertext $\mathsf{ct}_{\vec{\mathbf{u}}} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \vec{\mathbf{u}}; \mathsf{r}_{\vec{\mathbf{u}}})$.

[7] The arrowhead denotes that in this round messages are outgoing from party $\mathcal{P}^*$.

2. Compute $\pi_{\vec{\mathbf{u}}} \leftarrow \mathsf{Prove}(\mathsf{st}_{\vec{\mathbf{u}}}, \mathsf{wit}_{\vec{\mathbf{u}}})$ for $\mathsf{st}_{\vec{\mathbf{u}}} = (\mathsf{ct}_{\vec{\mathbf{u}}}, \mathsf{pk}^{\mathsf{TFHE}}) \in L_1$ using witness $\mathsf{wit}_{\vec{\mathbf{u}}} = (\vec{\mathbf{u}}, \mathsf{r}_{\vec{\mathbf{u}}})$ (language L_1 is defined in Figure 3).
3. Pick a set S consisting of t parties amongst $\mathcal{P}_1, \ldots, \mathcal{P}_n$. For simplicity, without loss of generality, we assume that $\mathcal{P}^*$ is also part of set S.
4. To each party $\mathcal{P}_i \in S$, send $(\mathsf{ct}_{\vec{\mathbf{u}}}, \pi_{\vec{\mathbf{u}}})$.

Statement: The statement st is as follows: $\mathsf{st} = (\mathsf{ct}, \mathsf{pk})$.

Witness: The witness wit is as follows: $\mathsf{wit} = (\mathsf{x}, \mathsf{r})$.

Relation: $\mathsf{R}_1(\mathsf{st}, \mathsf{wit}) = 1$ if and only if $\mathsf{ct} = \mathsf{TFHE.Enc}(\mathsf{pk}, \mathsf{x}; \mathsf{r})$.

Figure 3: NP language L_1

– **Round 2:** $(\rightarrow \mathcal{P}^*)$ Each party $\mathcal{P}_i \in S$ (except $\mathcal{P}^*$) does the following:

1. Abort and output $\perp$ if $\mathsf{Verify}(\pi_{\vec{\mathbf{u}}}, \mathsf{st}_{\vec{\mathbf{u}}}) \neq 1$ for language L_1 where the statement $\mathsf{st}_{\vec{\mathbf{u}}} = (\mathsf{ct}_{\vec{\mathbf{u}}}, \mathsf{pk}^{\mathsf{TFHE}})$.
2. Sample a uniformly random one-time key $K_i \leftarrow \{0,1\}^{\lambda}$ and compute $\mathsf{ct}_{K_i} = \mathsf{TFHE.Enc}\,(\mathsf{pk}^{\mathsf{TFHE}}, K_i; \mathsf{r}_{K_i})$.
3. Compute $\pi_{K_i} \leftarrow \mathsf{Prove}(\mathsf{st}_{K_i}, \mathsf{wit}_{K_i})$ for $\mathsf{st}_{K_i} = (\mathsf{ct}_{K_i}, \mathsf{pk}^{\mathsf{TFHE}}) \in L_1$ using the witness $\mathsf{wit}_{K_i} = (K_i, \mathsf{r}_{K_i})$ (language L_1 is defined in Figure 3).
4. Compute signatures $\sigma_{i,0} = \mathsf{Sign}(\mathsf{sk}_i, \mathsf{ct}_{\vec{\mathbf{u}}})$ and $\sigma_{i,1} = \mathsf{Sign}(\mathsf{sk}_i, \mathsf{ct}_{K_i})$.
5. Send the following to the party $\mathcal{P}^*$: $(\mathsf{ct}_{K_i}, \pi_{K_i}, \sigma_{i,0}, \sigma_{i,1})$.

– **Round 3:** $(\mathcal{P}^* \rightarrow)$ Party $\mathcal{P}^*$ checks if there exists some party $\mathcal{P}_i \in S$ such that $\mathsf{Verify}(\pi_{K_i}, \mathsf{st}_{K_i}) \neq 1$ for language L_1 where $\mathsf{st}_{K_i} = (\mathsf{ct}_{K_i}, \mathsf{pk}^{\mathsf{TFHE}})$. If yes, it outputs $\perp$ and aborts. Otherwise, it sends $\{(\mathsf{ct}_{K_i}, \pi_{K_i}, \sigma_{i,0}, \sigma_{i,1})\}_{\mathcal{P}_i \in S}$ to each party $\mathcal{P}_i \in S$.

– **Round 4:** $(\rightarrow \mathcal{P}^*)$ Each party $\mathcal{P}_i \in S$ (except $\mathcal{P}^*$) does the following:

1. If there exists some party $\mathcal{P}_j \in S$ such that $\mathsf{Verify}(\pi_{K_j}, \mathsf{st}_{K_j}) \neq 1$ for language L_1 where $\mathsf{st}_{K_j} = (\mathsf{ct}_{K_j}, \mathsf{pk}^{\mathsf{TFHE}})$ (OR) $\mathsf{Verify}(\mathsf{vk}_j, \mathsf{ct}_{\vec{\mathbf{u}}}, \sigma_{j,0}) \neq 1$ (OR) $\mathsf{Verify}(\mathsf{vk}_j, \mathsf{ct}_{K_j}, \sigma_{j,1})$ $\neq 1$, then output $\perp$ and abort.
2. Otherwise, for each $\mathcal{P}_j \in S$, do the following:
   – Compute $\mathsf{ct}_{\mathcal{C},j} = \mathsf{TFHE.Eval}(\mathsf{pk}^{\mathsf{TFHE}}, \mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\vec{\mathbf{w}}}, \mathsf{ct}_{\vec{\mathbf{u}}}, \mathsf{ct}_{K_j})$ using circuit $\mathcal{C}$ (Figure 4). Note that $\mathsf{ct}_{\mathcal{C},j}$ is either an encryption K_j or an encryption of 0^{λ}.

Inputs: A template $\vec{\mathbf{w}} \in \{0,1\}^{\ell}$, measurement $\vec{\mathbf{u}} \in \{0,1\}^{\ell}$ and string $K \in \{0,1\}^{\lambda}$.

Computation: If $\mathsf{Dist}(\vec{\mathbf{u}}, \vec{\mathbf{w}}) = 1$, output K. Else, output 0^{λ}.

Figure 4: Circuit $\mathcal{C}$

   – Compute a partial decryption: $\mu_{i,j} = \mathsf{TFHE.PartialDec}(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{ct}_{\mathcal{C},j})$.
   – Compute $\pi_{i,j} \leftarrow \mathsf{Prove}(\mathsf{st}_{i,j}, \mathsf{wit}_i)$ for $\mathsf{st}_{i,j} = (\mathsf{ct}_{\mathcal{C},j}, \mu_{i,j}, \mathsf{com}_i) \in L_2$ using $\mathsf{wit}_i = (\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{r}_i^{\mathsf{com}})$ (language L_2 is defined in Figure 5).

3. Compute partial signature $\mathsf{Token}_i = \mathsf{TS.Sign}(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{msg})$ and ciphertext $\mathsf{ct}_i = K_i \oplus \mathsf{Token}_i$.
4. Send $(\mathsf{ct}_i, \{(\pi_{i,j}, \mu_{i,j})\}_{\mathcal{P}_j \in S})$ to $\mathcal{P}^*$.

Statement: The statement st is as follows: $\mathsf{st} = (\mathsf{ct}, \mu, \mathsf{com})$.

Witness: The witness wit is as follows: $\mathsf{wit} = (\mathsf{sk}^{\mathsf{TFHE}}, \mathsf{r})$.

Relation: $\mathsf{R}_2(\mathsf{st}, \mathsf{wit}) = 1$ if and only if: (a) $\mathsf{TFHE.PartialDec}(\mathsf{sk}^{\mathsf{TFHE}}, \mathsf{ct}) = \mu$ *and* (b) $\mathsf{Commit}(\mathsf{sk}^{\mathsf{TFHE}}, \mathsf{r}) = \mathsf{com}$.

Figure 5: NP language L_2

– **Output Computation:** Every party $\mathcal{P}_i \in S$ outputs $(\mathsf{msg}, \mathcal{P}^*, S)$. Additionally, party $\mathcal{P}^*$ does the following to generate a token:

1. For each $\mathcal{P}_j \in S$, do the following:
   1. For each $\mathcal{P}_i \in S$, abort if $\mathsf{Verify}(\pi_{i,j}, \mathsf{st}_{i,j}) \neq 1$ for language L_2 where $\mathsf{st}_{i,j} = (\mathsf{ct}_{\mathcal{C},j}, \mu_{i,j}, \mathsf{com}_i)$.
   2. Set $K_j = \mathsf{TFHE.Combine}(\{\mu_{i,j}\}_{\mathcal{P}_i \in S})$. If $K_j = 0^\lambda$, output $\perp$.
   3. Otherwise, recover partial signature $\mathsf{Token}_j = K_j \oplus \mathsf{ct}_j$.
2. Reconstruct the signature as $\mathsf{Token} = \mathsf{TS.Combine}(\{\mathsf{Token}_i\}_{i \in S})$.
3. If $\mathsf{TS.Verify}(\mathsf{vk}^{\mathsf{TS}}, \mathsf{msg}, \mathsf{Token}) = 1$, then output $\{\mathsf{Token}_i\}_{\mathcal{P}_i \in S}$. Else, output $\perp$.

Token Verification: Given a verification key $\mathsf{vk}^{\mathsf{TS}}$, message msg and a set of partial tokens $\{\mathsf{Token}_i\}_{\mathcal{P}_i \in S}$, the token verification algorithm outputs 1 if $\mathsf{TS.Verify}(\mathsf{vk}^{\mathsf{TS}}, \mathsf{msg}, \mathsf{Token}) = 1$, where $\mathsf{Token} = \mathsf{TS.Combine}(\{\mathsf{Token}_i\}_{\mathcal{P}_i \in S})$.

7 Cosine Similarity: Single Corruption

In this section, we construct an efficient four round secure fuzzy threshold tokenizer in the Random Oracle (RO) model for Euclidean Distance and Cosine Similarity. Our protocol satisfies Definition 1 for any n with threshold $t = 3$ and is secure against a malicious adversary that can corrupt any one party. The special case of $n = 3$ corresponds to the popularly studied three party honest majority setting. We first focus on the Cosine Similarity distance measure. At the end of the section, we explain how to extend our result for Euclidean Distance. Formally:

Theorem 5. *Assuming unforgeable threshold signatures, two message OT in the CRS model, circuit-private additively homomorphic encryption and NIZKs for NP languages L_1, L_2 defined below, there exists a four round secure fuzzy threshold tokenizer protocol for Cosine Similarity. The protocol works for any n, threshold $t = 3$ and is secure against a malicious adversary that can corrupt any one party.*

We describe the construction below and defer the proof to Appendix B.3.

Paillier Encryption Scheme. The Paillier encryption scheme [Pai99] is an example of a circuit-private additively homomorphic encryption based on the N^{th} residuosity assumption. With respect to Paillier, we can also build NIZK arguments for languages L_1 and L_2 defined below, in the RO model. Formally:

Imported Theorem 1 ([DJ01])**.** *Assuming the hardness of the N^{th} residuosity assumption, there exists a NIZK for language L_1, defined below, in the RO model.*

Imported Theorem 2 ([CDN01])**.** *Assuming the hardness of the N^{th} residuosity assumption, there exists a NIZK for language L_2, defined below, in the RO model.*

The above NIZKs are very efficient and only require a constant number of group operations for both prover and verifier. Two message OT in the CRS model can be built assuming DDH/LWE/Quadratic Residuosity/N^{th} residuosity [NP01, PVW08, HK12]. Threshold signatures can be built assuming LWE/Gap-DDH/RSA [BGG+18, Bol03, Sho00]. Instantiating the primitives used in Theorem 5, we get the following corollary:

Corollary 6. *Assuming the hardness of the N^{th} residuosity assumption and LWE, there exists a four round secure fuzzy threshold tokenizer protocol for Cosine Similarity in the RO model. The protocol works for any n, $t = 3$ and is secure against a malicious adversary that can corrupt any one party.*

NP Languages.
Let $(\mathsf{AHE.Setup}, \mathsf{AHE.Enc}, \mathsf{AHE.Add}, \mathsf{AHE.ConstMul}, \mathsf{AHE.Dec})$ be an additively homomorphic encryption scheme. Let $\mathsf{epk} \leftarrow \mathsf{AHE.Setup}(1^\lambda)$, $m = \mathsf{poly}(\lambda)$.
Language L_1:
Statement: $\mathsf{st} = (\mathsf{ct}, \mathsf{pk})$. **Witness:** $\mathsf{wit} = (\mathsf{x}, \mathsf{r})$.
Relation: $\mathsf{R}_1(\mathsf{st}, \mathsf{wit}) = 1$ if $\mathsf{ct} = \mathsf{AHE.Enc}(\mathsf{epk}, \mathsf{x}; \mathsf{r})$ AND $x \in \{0,1\}^m$

Language L_2:
Statement: $\mathsf{st} = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{pk})$. **Witness:** $\mathsf{wit} = (\mathsf{x}_2, \mathsf{r}_2, \mathsf{r}_3)$.
Relation: $\mathsf{R}_2(\mathsf{st}, \mathsf{wit}) = 1$ if

$$\mathsf{ct}_2 = \mathsf{AHE.Enc}(\mathsf{epk}, \mathsf{x}_2; \mathsf{r}_2) \text{ AND } \mathsf{ct}_3 = \mathsf{AHE.ConstMul}(\mathsf{pk}, \mathsf{ct}_1, \mathsf{x}_2; \mathsf{r}_3).$$

7.1 Construction

Let RO denote a random oracle, d be the threshold value for Cosine Similarity. Recall that we denote $\mathsf{Dist}(\vec{\mathbf{u}}, \vec{\mathbf{w}}) = 1$ if $\mathsf{CS.Dist}(\vec{\mathbf{u}}, \vec{\mathbf{w}}) \geq d$. Let $(\mathsf{Share}, \mathsf{Recon})$ be a $(2, n)$ threshold secret sharing scheme, $\mathsf{TS} = (\mathsf{TS.Gen}, \mathsf{TS.Sign}, \mathsf{TS.Combine}, \mathsf{TS.Verify})$ be a threshold signature scheme, $(\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ denote a secret key encryption scheme, PRF denote a pseudorandom function, $(\mathsf{Garble}, \mathsf{Eval})$ denote a garbling scheme for circuits, $(\mathsf{Prove}, \mathsf{Verify})$ be a NIZK system in the RO model, $\mathsf{AHE} = (\mathsf{AHE.Setup}, \mathsf{AHE.Enc}, \mathsf{AHE.Add}, \mathsf{AHE.ConstMul}, \mathsf{AHE.Dec})$ denote a circuit-private additively homomorphic encryption scheme and $\mathsf{OT} = (\mathsf{OT.Setup}, \mathsf{OT.Round}_1, \mathsf{OT.Round}_2, \mathsf{OT.Output})$ be a two message oblivious transfer protocol in the CRS model.

We now describe the construction of our four round secure fuzzy threshold tokenizer protocol π^{CS} for Cosine Similarity.

Setup: The trusted authority does the following:

– Compute $(\mathsf{pp}^{\mathsf{TS}}, \mathsf{vk}^{\mathsf{TS}}, \mathsf{sk}_1^{\mathsf{TS}}, \ldots, \mathsf{sk}_n^{\mathsf{TS}}) \leftarrow \mathsf{TS.Gen}(1^\lambda, n, k)$.

– For $i \in [n]$, generate $\mathsf{crs}_i \leftarrow \mathsf{OT.Setup}(1^\lambda)$ and pick a random PRF key k_i.

– For $i \in [n]$, give $(\mathsf{pp}^{\mathsf{TS}}, \mathsf{vk}^{\mathsf{TS}}, \mathsf{sk}_i^{\mathsf{TS}}, \{\mathsf{crs}_j\}_{j\in[n]}, \{\mathsf{k}_j\}_{j\in[n]\setminus i})$ to party $\mathcal{P}_i$.

Enrollment: In this phase, any party $\mathcal{P}_i$ that wishes to enroll, queries the trusted authority which then does the following:

– Sample a random vector $\overrightarrow{\mathbf{w}}$ from the distribution $\mathcal{W}$. Without loss of generality, let's assume that the $\mathsf{L2}$-norm of $\overrightarrow{\mathbf{w}}$ is 1.

– For each $i \in [n]$, do the following:

  – Compute $(\overrightarrow{\mathbf{w}}_i, \overrightarrow{\mathbf{v}}_i) \leftarrow \mathsf{Share}(1^\lambda, \overrightarrow{\mathbf{w}}, n, 2)$.
  – Compute $(\mathsf{esk}_i, \mathsf{epk}_i) \leftarrow \mathsf{AHE.Setup}(1^\lambda)$.
  – Let $\overrightarrow{\mathbf{w}}_i = (\mathsf{w}_{i,1}, \ldots, \mathsf{w}_{i,\ell})$. $\forall j \in [\ell]$, compute $[\![\mathsf{w}_{i,j}]\!] = \mathsf{AHE.Enc}(\mathsf{epk}_i, \mathsf{w}_{i,j})$.
  – Give $(\overrightarrow{\mathbf{w}}_i, \mathsf{sk}_i, \mathsf{pk}_i, \{[\![\mathsf{w}_{i,j}]\!]\}_{j\in[\ell]})$ to party $\mathcal{P}_i$ and $(\overrightarrow{\mathbf{v}}_i, \mathsf{pk}_i, \{[\![\mathsf{w}_{i,j}]\!]\}_{j\in[\ell]})$ to all the other parties.

SignOn Phase: In the SignOn phase, let's consider party $\mathcal{P}_i$ that uses an input vector $\overrightarrow{\mathbf{u}} = (\mathsf{u}_1, \ldots, \mathsf{u}_\ell)$ and a message msg on which it wants a token. $\mathcal{P}_i$ picks two other parties $\mathcal{P}_j$ and $\mathcal{P}_k$ and interacts with them in the below protocol.

Round 1: $(\mathcal{P}_i \rightarrow)$ [8] Party $\mathcal{P}_i$ does the following:

1. Let $S = (\mathcal{P}_j, \mathcal{P}_k)$ with $j < k$.

2. For each $j \in [\ell]$, compute the following:

   – $[\![\mathsf{u}_j]\!] = \mathsf{AHE.Enc}(\mathsf{epk}_i, \mathsf{u}_j; \mathsf{r}_{1,j})$. $\pi_{1,j} \leftarrow \mathsf{Prove}(\mathsf{st}_{1,j}, \mathsf{wit}_{1,j})$ for $\mathsf{st}_{1,j} = ([\![\mathsf{u}_j]\!], \mathsf{epk}_i) \in L_1$ using $\mathsf{wit}_{1,j} = (\mathsf{u}_j, \mathsf{r}_{1,j})$.

   – $[\![\mathsf{u}_j^2]\!] = \mathsf{AHE.ConstMul}(\mathsf{epk}_i, [\![\mathsf{u}_j]\!], \mathsf{u}_j; \mathsf{r}_{2,j})$. $\pi_{2,j} \leftarrow \mathsf{Prove}(\mathsf{st}_{2,j}, \mathsf{wit}_{2,j})$ for $\mathsf{st}_{2,j} = ([\![\mathsf{u}_j]\!], [\![\mathsf{u}_j]\!], [\![\mathsf{u}_j^2]\!], \mathsf{epk}_i) \in L_2$ using $\mathsf{wit}_{2,j} = (\mathsf{u}_j, \mathsf{r}_{1,j}, \mathsf{r}_{2,j})$.

   – $[\![\mathsf{w}_{i,j} \cdot \mathsf{u}_j]\!] = \mathsf{AHE.ConstMul}(\mathsf{epk}_i, [\![\mathsf{w}_{i,j}]\!], \mathsf{u}_j; \mathsf{r}_{3,j})$. $\pi_{3,j} \leftarrow \mathsf{Prove}(\mathsf{st}_{3,j}, \mathsf{wit}_{3,j})$ for $\mathsf{st}_{3,j} = ([\![\mathsf{w}_{i,j}]\!], [\![\mathsf{u}_j]\!], [\![\mathsf{w}_{i,j} \cdot \mathsf{u}_j]\!], \mathsf{epk}_i) \in L_2$ using $\mathsf{wit}_{3,j} = (\mathsf{u}_j, \mathsf{r}_{1,j}, \mathsf{r}_{3,j})$.

3. To both parties in S, send $\mathsf{msg}_1 = (S, \mathsf{msg}, \{[\![\mathsf{u}_j]\!], [\![\mathsf{u}_j^2]\!], [\![\mathsf{w}_{i,j} \cdot \mathsf{u}_j]\!], \pi_{1,j}, \pi_{2,j}, \pi_{3,j}\}_{j\in[\ell]})$.

Round 2: $(\rightarrow \mathcal{P}_i)$ Both parties $\mathcal{P}_j$ and $\mathcal{P}_k$ do the following:

1. Abort if any of the proofs $\{\pi_{1,j}, \pi_{2,j}, \pi_{3,j}\}_{j\in[\ell]}$ don't verify.

2. Generate randomness $(\mathsf{a}, \mathsf{b}, \mathsf{e}, \mathsf{f}, \mathsf{p}, \mathsf{q}, \mathsf{r_z}) \leftarrow \mathsf{PRF}(\mathsf{k}_i, \mathsf{msg}_1)$.

3. Using the algorithms of AHE, compute $[\![\mathsf{x}_1]\!], [\![\mathsf{x}_2]\!], [\![\mathsf{y}_1]\!], [\![\mathsf{y}_2]\!], [\![\mathsf{z}_1]\!], [\![\mathsf{z}_2]\!]$ as follows:

   – $\mathsf{x}_1 = \langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}_i \rangle$, $\mathsf{y}_1 = \langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{u}} \rangle$, $\mathsf{z}_1 = (\langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{v}}_i \rangle + \mathsf{r_z})$.
   – $\mathsf{x}_2 = (\mathsf{a} \cdot \mathsf{x}_1 + \mathsf{b})$, $\mathsf{y}_2 = (\mathsf{e} \cdot \mathsf{y}_1 + \mathsf{f})$, $\mathsf{z}_2 = (\mathsf{p} \cdot \mathsf{z}_1 + \mathsf{q})$

4. Send $([\![\mathsf{x}_2]\!], [\![\mathsf{y}_2]\!], [\![\mathsf{z}_1]\!], [\![\mathsf{z}_2]\!])$ to $\mathcal{P}_i$.

[8] The arrowhead denotes that in this round messages are outgoing from party $\mathcal{P}_i$.

Round 3: $(\mathcal{P}_i \rightarrow\ )$ Party $\mathcal{P}_i$ does the following:

1. Abort if the tuples sent by both $\mathcal{P}_j$ and $\mathcal{P}_k$ in round 2 were not the same.
2. Compute $\mathsf{x}_1 = \langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}_i \rangle$, $\mathsf{x}_2 = \mathsf{AHE.Dec}(\mathsf{esk}_i, [\![\mathsf{x}_2]\!])$.
3. Compute $\mathsf{y}_1 = \langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{u}} \rangle$, $\mathsf{y}_2 = \mathsf{AHE.Dec}(\mathsf{esk}_i, [\![\mathsf{y}_2]\!])$.
4. Compute $\mathsf{z}_1 = \mathsf{AHE.Dec}(\mathsf{esk}_i, [\![\mathsf{z}_1]\!])$, $\mathsf{z}_2 = \mathsf{AHE.Dec}(\mathsf{esk}_i, [\![\mathsf{z}_2]\!])$.
5. Generate and send $\mathsf{msg}_3 = \{\mathsf{ot}^{\mathsf{rec}}_{\mathsf{s,t}} \leftarrow \mathsf{OT.Round}_1(\mathsf{crs}_i, \mathsf{s}_\mathsf{t})\}_{\mathsf{s}\in\{\mathsf{x,y,z}\},\mathsf{t}\in\{1,2\}}$.

Round 4: $(\mathcal{P}_j \rightarrow \mathcal{P}_i)$ Party $\mathcal{P}_j$ does the following:

1. Compute $\widetilde{\mathcal{C}} = \mathsf{Garble}(\mathcal{C})$ for the circuit $\mathcal{C}$ described in Figure 6.
2. For each $\mathsf{s} \in \{\mathsf{x,y,z}\}, \mathsf{t} \in \{0,1\}$, let $\mathsf{lab}^0_{\mathsf{s,t}}, \mathsf{lab}^1_{\mathsf{s,t}}$ denote the labels of the garbled circuit $\widetilde{\mathcal{C}}$ corresponding to input wires s_t. Generate $\mathsf{ot}^{\mathsf{sen}}_{\mathsf{s,t}} = \mathsf{OT.Round}_2(\mathsf{crs}_i, \mathsf{lab}^0_{\mathsf{s,t}}, \mathsf{lab}^1_{\mathsf{s,t}}, \mathsf{ot}^{\mathsf{rec}}_{\mathsf{s,t}})$. Let $\mathsf{ot}^{\mathsf{sen}} = \{\mathsf{ot}^{\mathsf{sen}}_{\mathsf{s,t}}\}_{\mathsf{s}\in\{\mathsf{x,y,z}\},\mathsf{t}\in\{1,2\}}$
3. Compute $\mathsf{pad} = \mathsf{PRF}(\mathsf{k}_i, \mathsf{msg}_3)$. Set $\mathsf{ct}_j = \mathsf{SKE.Enc}(\mathsf{pad}, \mathsf{TS.Sign}(\mathsf{sk}^{\mathsf{TS}}_j, \mathsf{msg}))$.
4. Send $(\widetilde{\mathcal{C}}, \mathsf{ot}^{\mathsf{sen}}, \mathsf{ct}_j)$ to $\mathcal{P}_i$.

Round 4: $(\mathcal{P}_k \rightarrow \mathcal{P}_i)$ Party $\mathcal{P}_k$ does the following:

1. Compute $(\widetilde{\mathcal{C}}, \mathsf{ot}^{\mathsf{sen}}, \mathsf{pad})$ exactly as done by $\mathcal{P}_j$.
2. Set $\mathsf{ct}_k = \mathsf{SKE.Enc}(\mathsf{pad}, \mathsf{TS.Sign}(\mathsf{sk}^{\mathsf{TS}}_k, \mathsf{msg}))$.
3. Send $(\mathsf{RO}(\widetilde{\mathcal{C}}, \mathsf{ot}^{\mathsf{sen}}), \mathsf{ct}_k)$ to $\mathcal{P}_i$.

Output Computation: Parties $\mathcal{P}_j, \mathcal{P}_k$ output $(\mathsf{msg}, \mathcal{P}_i, S)$. Party $\mathcal{P}_i$ does:

1. Let $(\widetilde{\mathcal{C}}, \mathsf{ot}^{\mathsf{sen}}, \mathsf{ct}_j)$ be the message received from $\mathcal{P}_j$ and $(\mathsf{msg}_4, \mathsf{ct}_k)$ be the message received from $\mathcal{P}_k$. Abort if $\mathsf{RO}(\widetilde{\mathcal{C}}, \mathsf{ot}^{\mathsf{sen}}) \neq \mathsf{msg}_4$.
2. For each $\mathsf{s} \in \{\mathsf{x,y,z}\}, \mathsf{t} \in \{0,1\}$, compute $\mathsf{lab}_{\mathsf{s,t}} = \mathsf{OT.Output}(\mathsf{ot}^{\mathsf{sen}}_{\mathsf{s,t}}, \mathsf{ot}^{\mathsf{rec}}_{\mathsf{s,t}}, \mathsf{r}^{\mathsf{ot}}_{\mathsf{s,t}})$. Let $\mathsf{lab} = \{\mathsf{lab}_{\mathsf{s,t}}\}_{\mathsf{s}\in\{\mathsf{x,y,z}\},\mathsf{t}\in\{0,1\}}$. Compute $\mathsf{pad} = \mathsf{Eval}(\widetilde{\mathcal{C}}, \mathsf{lab})$.
3. Compute $\mathsf{Token}_j = \mathsf{SKE.Dec}(\mathsf{pad}, \mathsf{ct}_j), \mathsf{Token}_k = \mathsf{SKE.Dec}(\mathsf{pad}, \mathsf{ct}_k), \mathsf{Token}_i = \mathsf{TS.Sign}(\mathsf{sk}^{\mathsf{TS}}_i,$ $\mathsf{msg})$, $\mathsf{Token} \leftarrow \mathsf{TS.Combine}(\{\mathsf{Token}_s\}_{s\in\{i,j,k\}})$.
4. Output $\{\mathsf{Token}_s\}_{s\in\{i,j,k\}}$ if $\mathsf{TS.Verify}(\mathsf{vk}^{\mathsf{TS}}, \mathsf{msg}, \mathsf{Token})$. Else, output $\perp$.

Token Verification: Given a verification key $\mathsf{vk}^{\mathsf{TS}}$, message msg and token $(\mathsf{Token}_i, \mathsf{Token}_j, \mathsf{Token}_k)$, the token verification algorithm does the following:

1. Compute $\mathsf{Token} \leftarrow \mathsf{TS.Combine}(\{\mathsf{Token}_s\}_{s\in\{i,j,k\}})$.
2. Output 1 if $\mathsf{TS.Verify}(\mathsf{vk}^{\mathsf{TS}}, \mathsf{msg}, \mathsf{Token}) = 1$. Else, output 0.

Inputs: $(\mathsf{x}_1, \mathsf{x}_2, \mathsf{y}_1, \mathsf{y}_2, \mathsf{z}_1, \mathsf{z}_2)$. **Hardwired values:** $(\mathsf{a}, \mathsf{b}, \mathsf{e}, \mathsf{f}, \mathsf{p}, \mathsf{q}, \mathsf{r_z}, \mathsf{pad}, d^2)$. **Computation:**

- Abort if $\mathsf{x}_2 \neq (\mathsf{a} \cdot \mathsf{x}_1 + \mathsf{b})$ (or) $\mathsf{y}_2 \neq (\mathsf{e} \cdot \mathsf{y}_1 + \mathsf{f})$ (or) $\mathsf{z}_2 \neq (\mathsf{p} \cdot \mathsf{z}_1 + \mathsf{q})$
- Compute $\mathsf{IP} = (\mathsf{z}_1 - \mathsf{r_z}) + \mathsf{x}_1$
- If $\mathsf{IP}^2 \geq (d^2 \cdot \mathsf{y}_1)$, output pad. Else, output $\perp$.

Figure 6: Circuit $\mathcal{C}$ to be garbled.

7.2 Euclidean Distance

Recall that given two vectors $\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}$, the square of the Euclidean Distance $\mathsf{EC.Dist}$ between them relates to their Cosine Similarity $\mathsf{CS.Dist}$ as follows: $\mathsf{EC.Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}) = (\langle \overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{u}} \rangle + \langle \overrightarrow{\mathbf{w}}, \overrightarrow{\mathbf{w}} \rangle - 2 \cdot \mathsf{CS.Dist}(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}))$. Thus, it is easy to observe that the above protocol and analysis easily extends for Euclidean Distance too.

8 Open Problems

We list some interesting open problems for future work.

- Can we define game-based security definitions and design more efficient protocols for FTT that satisfy those? Typically, game-based definitions are weaker than simulation-based ones and hence easier to design more efficient protocols for.
- Can we design protocols that tolerate adaptive corruptions of the parties by the adversary? Currently, all our protocols are secure only against an adversary that performs static corruptions before the protocol begins.
- Can we design a definition and protocols that allow parties to dynamically join and leave the system? This captures the setting where users or enterprises add or remove new devices. Our current framework only captures a static setting with an apriori fixed number of parties and there are several challenges on the definitional front and in protocol design to capture this. For instance, it is not even clear how the threshold signatures would work once more parties join.
- Can we design protocols that allow for rotation of signature keys over time as is common in real world systems?

References

[AMMM18] Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. PASTA: PASsword-based threshold authentication. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 2042–2059. ACM Press, October 2018.

[AMMR18] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed symmetric-key encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1993–2010. ACM Press, October 2018.

[appa] About Face ID advanced technology. https://support.apple.com/en-us/HT208108. Accessed on June 7, 2020.

[appb] iOS Security — iOS 12. https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf. Page-8, Accessed on June 7, 2020.

[BCP13] Julien Bringer, Hervé Chabanne, and Alain Patey. SHADE: Secure HAmming DistancE computation from oblivious transfer. In Andrew A. Adams, Michael Brenner, and Matthew Smith, editors, *FC 2013 Workshops*, LNCS, pages 164–176. Springer, Heidelberg, April 2013.

[BCV16] Olivier Blazy, Céline Chevalier, and Damien Vergnaud. Mitigating server breaches in password-based authentication: Secure and efficient solutions. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 3–18. Springer, Heidelberg, February / March 2016.

[BDCG13] Carlo Blundo, Emiliano De Cristofaro, and Paolo Gasti. Espresso: Efficient privacy-preserving evaluation of sample set similarity. In Roberto Di Pietro, Javier Herranz, Ernesto Damiani, and Radu State, editors, *Data Privacy Management and Autonomous Spontaneous Security*, 2013.

[BFH$^+$19] Carsten Baum, Tore K. Frederiksen, Julia Hesse, Anja Lehmann, and Avishay Yanai. Proactively secure distributed single sign-on, or how to trust a hacked server. Cryptology ePrint Archive, Report 2019/1470, 2019. https://eprint.iacr.org/2019/1470.

[BGG$^+$18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, August 2018.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Heidelberg, April 2015.

[BGI$^+$18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018*, volume 94, pages 21:1–21:21. LIPIcs, January 2018.

[BIK17a] Amos Beimel, Yuval Ishai, and Eyal Kushilevitz. Ad hoc PSM protocols: Secure computation without coordination. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 580–608. Springer, Heidelberg, April / May 2017.

[BIK$^+$17b] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1175–1191. ACM Press, October / November 2017.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Heidelberg, April / May 2018.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Heidelberg, January 2003.

[Boy04] Xavier Boyen. Reusable cryptographic fuzzy extractors. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 2004*, pages 82–91. ACM Press, October 2004.

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 190–213. Springer, Heidelberg, August 2016.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[Can04] Ran Canetti. Universally Composable Signature, Certification, and Authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219, 2004.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2015.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Heidelberg, May 2001.

[CHK$^+$05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005.

[CSS12] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Privacy-preserving stream aggregation with fault tolerance. In Angelos D. Keromytis, editor, *FC 2012*, volume 7397 of *LNCS*, pages 200–214. Springer, Heidelberg, February / March 2012.

[DHP+18] Pierre-Alain Dupont, Julia Hesse, David Pointcheval, Leonid Reyzin, and Sophia Yakoubov. Fuzzy password-authenticated key exchange. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 393–424. Springer, Heidelberg, April / May 2018.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Heidelberg, February 2001.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, Heidelberg, May 2004.

[DSB17] Trung Dinh, Ron Steinfeld, and Nandita Bhattacharjee. A lattice-based approach to privacy-preserving biometric authentication without relying on trusted third parties. In *ISPEC*, 2017.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.

[ess] Advantages and disadvantages of biometrics. https://www.ukessays.com/dissertation/examples/information-systems/advantages-and-disadvantages-of-biometrics.php?vref=1. Accessed on June 7, 2020.

[fid] FIDO Alliance. https://fidoalliance.org/. Accessed on June 7, 2020.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994.

[GFN+17] Paul A Grassi, JL Fenton, EM Newton, RA Perlner, AR Regenscheid, WE Burr, JP Richer, NB Lefkovitz, JM Danker, YY Choong, KK Greene, and MF Theofanos. Nist special publication 800-63b: Digital identity guidelines: Authentication and lifecycle management. June 2017.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499. Springer, Heidelberg, April / May 2018.

[HK12] Shai Halevi and Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. *Journal of Cryptology*, 25(1):158–193, January 2012.

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *ISTCS '97*, Washington, DC, USA, 1997.

[JL13] Marc Joye and Benoît Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In Ahmad-Reza Sadeghi, editor, *FC 2013*, volume 7859 of *LNCS*, pages 111–125. Springer, Heidelberg, April 2013.

[KZ08] Aggelos Kiayias and Hong-Sheng Zhou. Equivocal blind signatures and adaptive UC-security. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 340–355. Springer, Heidelberg, March 2008.

[lis] List of data breaches. https://en.wikipedia.org/wiki/List_of_data_breaches. Accessed on June 7, 2020.

[LWY$^+$] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *CVPR*.

[MRZ15] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 591–602. ACM Press, October 2015.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.

[nis] NISTIR Draft on Ongoing Face Recognition Vendor Test Part 1: Verification. https://pages.nist.gov/frvt/reports/11/frvt_report_2020_01_21.pdf. Accessed on June 7, 2020.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

[pix] Google Pixel Fingerprint. https://support.google.com/pixelphone/answer/6285273?hl=en. Accessed on June 7, 2020.

[prc] Privacy Rights Clearinghouse – Data Breaches. https://www.privacyrights.org/data-breaches. Accessed on June 7, 2020.

[PS16] Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 217–238. Springer, Heidelberg, October / November 2016.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Heidelberg, August 2019.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[sam] Samsung Galaxy: Iris Scans for Security. https://www.samsung.com/global/galaxy/galaxy-s8/security/. Accessed on June 7, 2020.

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Heidelberg, May 2000.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, 2015.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.

[w3s] Web Authentication: W3 Standard. https://www.w3.org/TR/2018/CR-webauthn-20180320/. Accessed on June 7, 2020.

[whi] White-Box Competition. https://whibox-contest.github.io/. Accessed on June 7, 2020.

[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 *CVPR*, 2018.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Cryptographic Definitions

A.1 Basic primitives

Definition 6. ***(Pseudorandom Function).*** *A pseudorandom function (PRF) is a polynomial-time computable function*

$$F : \{0,1\}^\lambda \times \{0,1\}^\ell \longrightarrow \{0,1\}^{\ell'},$$

such that for all PPT algorithms $\mathcal{A}$, we have

$$\left|\Pr\left[\mathcal{A}^{F(K,\cdot)} = 1\right] - \Pr\left[\mathcal{A}^{G(\cdot)} = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where $K \xleftarrow{R} \{0,1\}^\lambda$ and G is uniformly sampled from the set of all functions that map $\{0,1\}^\ell$ to $\{0,1\}^{\ell'}$.

Definition 7. ***(Symmetric-Key Encryption).*** *A symmetric-key encryption scheme* SKE *consists of the following polynomial-time algorithms:*

- $\mathsf{SKE}, \mathsf{Gen}(\lambda)$*: A probabilistic algorithm that takes the security parameter* λ *as input and outputs a secret-key* sk.
- $\mathsf{SKE.Enc}(\mathsf{sk}, \mathsf{m})$*: A probabilistic algorithm that takes as input a key* sk *and a plaintext* m*. Outputs a ciphertext* ct.
- $\mathsf{SKE.Dec}(\mathsf{sk}, \mathsf{ct})$*: A deterministic algorithm that takes as input a key* sk *and a ciphertext* ct*. Outputs the decrypted plaintext* m'.

The following correctness and security properties should be satisfied:

- ***Correctness.*** *For any message* m*, letting* $\mathsf{sk} = \mathsf{SKE}, \mathsf{Gen}(\lambda)$*, we have with overwhelmingly large probability*
$$\mathsf{SKE.Dec}(\mathsf{sk}, \mathsf{SKE.Enc}(\mathsf{sk}, \mathsf{m})) = \mathsf{m}.$$

- ***Security.*** *A symmetric-key encryption scheme* SKE *is said to be CPA-secure if for all PPT algorithms* $\mathcal{A}$ *and any two* arbitrary *plaintext messages* m_0 *and* m_1*, we have*
$$|\Pr[\mathcal{A}((\mathsf{m}_0, \mathsf{m}_1), \mathsf{ct}_0) = 1] - \Pr[\mathcal{A}((\mathsf{m}_0, \mathsf{m}_1), \mathsf{ct}_1) = 1]| \leq negl(\lambda),$$
where $sk = \mathsf{SKE}, \mathsf{Gen}(\lambda)$ *and* $\mathsf{ct}_b = \mathsf{SKE.Enc}(\mathsf{sk}, \mathsf{m}_b)$ *for* $b \in \{0, 1\}$.

Definition 8. ***(Public-Key Encryption).*** *A public-key encryption scheme* PKE *consists of the following polynomial-time algorithms:*

- $\mathsf{PKE.Gen}(\lambda)$*: A probabilistic algorithm that takes the security parameter* λ *as input and outputs a public key* pk *and a secret key* sk.
- $\mathsf{SKE.Enc}(\mathsf{pk}, \mathsf{m})$*: A probabilistic algorithm that takes as input a public key* pk *and a plaintext* m*. Outputs a ciphertext* ct.
- $\mathsf{SKE.Dec}(\mathsf{sk}, \mathsf{ct})$*: A deterministic algorithm that takes as input a key* sk *and a ciphertext* ct*. Outputs the decrypted plaintext* m'.

The following correctness and security properties should be satisfied:

- ***Correctness.*** *For any message* m*, letting* $(\mathsf{pk}, \mathsf{sk}) = \mathsf{PKE.Gen}(\lambda)$*, we have with overwhelmingly large probability*
$$\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{m})) = \mathsf{m}.$$

- ***Security.*** *A public-key encryption scheme is said to be CPA-secure if for all PPT algorithms* $\mathcal{A}$ *and any two* arbitrary *plaintext messages* m_0 *and* m_1*, we have*
$$|\Pr[\mathcal{A}(\mathsf{pk}, (\mathsf{m}_0, \mathsf{m}_1), \mathsf{ct}_0) = 1] - \Pr[\mathcal{A}(\mathsf{pk}, (\mathsf{m}_0, \mathsf{m}_1), \mathsf{ct}_1) = 1]| \leq negl(\lambda),$$
where $(\mathsf{pk}, \mathsf{sk}) = \mathsf{PKE.Gen}(\lambda)$ *and* $\mathsf{ct}_b = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{m}_b)$ *for* $b \in \{0, 1\}$.

Definition 9. ***(Garbled Circuits).*** *A garbling scheme for a class of circuits* $\mathcal{C}$ *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* $\hat{C}$ *and a set of labels* $\overleftarrow{\ell} = \{\ell_{j,0}, \ell_{j,1}\}_{j\in[n]}$.

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

The following correctness and security properties should be satisfied:

- ***Correctness.*** *For any circuit* $C \in \mathcal{C}$ *and any string* $(x_1, \dots, x_n) \in \{0,1\}^n$*, if* $(\hat{C}, \{\ell_{j,0}, \ell_{j,1}\}_{j\in[n]}) = \mathsf{Garble}(C)$*, we have*

$$\mathsf{Eval}((x_1, \dots, x_n), \{\ell_{j,x_j}\}_{j\in[n]}, \hat{C}) = C(x_1, \dots, x_n).$$

- ***Security.*** *There exists a PPT algorithm* Sim *such that for any circuit* $C \in \mathcal{C}$ *and any string* $(x_1, \dots, x_n) \in \{0,1\}^n$*, letting* $(\hat{C}, \{\ell_{j,0}, \ell_{j,1}\}_{j\in[n]}) = \mathsf{Garble}(C)$*, it holds that the ensembles*

$$(\hat{C}, \{\ell_{j,x_j}\}_{j\in[n]}) \quad \textit{and} \quad \mathsf{Sim}(1^\lambda, C(x))$$

are computationally indistinguishable.

Definition 10. ***(Non-Interactive Commitments).*** *A non-interactive commitment scheme is a probabilistic polynomial-time algorithm* Commit *that takes as input a message* m *together with random coins* $r \in \{0,1\}^\lambda$*, and returns a commitment* com*. The opening of a commitment* com *consists of strings* (m, r) *such that* $\mathsf{com} = \mathsf{Commit}(\mathsf{m}; r)$*. As for security, commitment schemes should satisfy two properties called hiding and binding:*

- ***Hiding:*** *A commitment scheme is perfectly (resp., computationally or statistically) hiding, if for all* $\mathsf{m}_0, \mathsf{m}_1$*, it holds that the ensembles*

$$\{\mathsf{Commit}(\mathsf{m}_0; r_0)\}_{r_0 \leftarrow \{0,1\}^\lambda} \quad \textit{and} \quad \{\mathsf{Commit}(\mathsf{m}_1; r_1)\}_{r_1 \leftarrow \{0,1\}^\lambda}$$

are identically distributed (resp., computationally or statistically close).

- ***Binding:*** *A commitment scheme is computationally (resp., perfectly) binding, if for all PPT (resp., computationally unbounded) adversaries* $\mathcal{A}$ *we have*

$$\Pr\left[\mathcal{A}(1^\lambda) = ((\mathsf{m}_0, r_0), (\mathsf{m}_1, r_1))\right] \leq negl(\lambda),$$

whenever $\mathsf{Commit}(\mathsf{m}_0; r_0) = \mathsf{Commit}(\mathsf{m}_1; r_1)$.

Definition 11. ***(Two-Message Oblivious Transfer.)*** *A two-message oblivious transfer (OT) protocol in the common reference string (CRS) model is a tuple* $\mathsf{OT} = (\mathsf{OT.Setup}, \mathsf{OT.Round}_1, \mathsf{OT.Round}_2, \mathsf{OT.Output})$ *defined as follows:*

- $\mathsf{OT.Setup}(1^\lambda)$*: A PPT algorithm that, given the security parameter* λ*, outputs a common reference string* crs.

- $\mathsf{OT.Round}_1(\mathsf{crs}, \beta)$*: A PPT algorithm that, given* crs *and a bit* $\beta \in \{0,1\}$*, outputs a message* m_1 *and a secret state* st.

- $\mathsf{OT.Round}_2(\mathsf{crs}, (\mu_0, \mu_1), \mathsf{m}_1)$*: A PPT algorithm that, given* crs*, a pair of strings* $\mu_0, \mu_1 \in \{0,1\}^\ell$ *(where* ℓ *is a parameter of the scheme) and a message* m_1*, outputs a message* m_2*.*

- $\mathsf{OT.Output}(\mathsf{crs}, \mathsf{st}, \beta, \mathsf{m}_2)$*: A deterministic algorithm that, given* crs*, a secret state* st*, a bit* $\beta \in \{0,1\}$ *and a message* m_2*, it outputs a string* $\mu' \in \{0,1\}^\ell$*.*

The following correctness and security properties should be satisfied:

- ***Correctness.*** *For any* $\lambda \in \mathbb{N}$*, any bit* $\beta \in \{0,1\}$*, and any pair of strings* $\mu_0, \mu_1 \in \{0,1\}^\ell$*, letting* $\mathsf{crs} = \mathsf{OT.Setup}(1^\lambda)$*,* $(\mathsf{m}_1, \mathsf{st}) = \mathsf{OT.Round}_1(\mathsf{crs}, \beta)$*,* $\mathsf{m}_2 = \mathsf{OT.Round}_2(\mathsf{crs}, (\mu_0, \mu_1), \mathsf{msg}_1)$ *and* $\mu' = \mathsf{OT.Output}(\mathsf{crs}, \mathsf{st}, \beta, \mathsf{m}_2)$*, we have* $\mu' = \mu$ *with overwhelmingly large probability.*

- ***Receiver Privacy.*** *For any* $\lambda \in \mathbb{N}$ *and for any PPT adversary* $\mathcal{A}$*, letting* $\mathsf{crs} = \mathsf{OT.Setup}(1^\lambda)$*, we have*

$$|\Pr[\mathcal{A}(\mathsf{crs}, \mathsf{OT.Round}_1(\mathsf{crs}, 0)) = 0] - \Pr[\mathcal{A}(\mathsf{crs}, \mathsf{OT.Round}_1(\mathsf{crs}, 1) = 0]| \le negl(\lambda),$$

where the probabilities are defined over the random coins used internally by the $\mathsf{OT.Round}_1$ *algorithm.*

- ***Sender Privacy****. For any* $\lambda \in \mathbb{N}$*, any* $\mathsf{m}_1 = \mathsf{m}_1(\lambda)$ *and any* $(\mu_0, \mu_1) = (\mu_0(\lambda), \mu_1(\lambda))$*, there exists a PPT simulator* $\mathsf{Sim} = (\mathsf{OT.Sim.Setup}, \mathsf{OT.Sim.Round}_2)$ *such that letting* $\mathsf{crs} = \mathsf{OT.Setup}(1^\lambda)$*,* $(\widehat{\mathsf{crs}}, \tau) = \mathsf{OT.Sim.Setup}(1^\lambda)$*, and* $\beta = \mathsf{OT.Sim.Round}_2(\tau, \mathsf{m}_1)$*, the following holds for any PPT adversary* $\mathcal{A}$*:*

$$|\Pr[\mathcal{A}(\mathsf{crs}, \mathsf{m}_1, \mathsf{m}_2) = 0] - \Pr[\mathcal{A}(\widehat{\mathsf{crs}}, \mathsf{m}_1, \widehat{\mathsf{m}_{2,\beta}}) = 0]| \le negl(\lambda),$$

where

$$\mathsf{m}_2 = \mathsf{OT.Round}_2(\mathsf{crs}, (\mu_0, \mu_1), \mathsf{m}_1) \quad , \quad \widehat{\mathsf{m}_{2,\beta}} = \mathsf{OT.Round}_2(\widehat{\mathsf{crs}}, (\mu_\beta, \mu_\beta), \mathsf{m}_1).$$

Definition 12. ***(Non-Interactive Zero-Knowledge.)*** *Let* R *be a relation and let the language* L *be a set of statements* $\{\mathsf{st} \in \{0,1\}^n\}$ *such that for each statement* $\mathsf{st} \in L$*, there exists a corresponding witness* wit *such that* $(\mathsf{st}, \mathsf{wit}) \in R$*. A non-interactive zero-knowledge (NIZK) proof system for* R *is a tuple of PPT algorithms* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{Prove}, \mathsf{Verify})$ *defined as follows:*

- $\mathsf{NIZK.Setup}(1^\lambda, 1^n)$*: A PPT algorithm that, given the security parameter* λ *and the statement length parameter* n*, outputs a common random string* crs*.*

- $\mathsf{Prove}(\mathsf{crs}, (\mathsf{st}, \mathsf{wit}))$*: A PPT algorithm that, given* crs*, a statement* $\mathsf{st} \in \{0,1\}^n$ *and a witness* wit *such that* $(\mathsf{st}, \mathsf{wit}) \in R$*, outputs a proof* π*.*

- $\mathsf{Verify}(\mathsf{crs}, \mathsf{st}, \pi)$*: A deterministic algorithm that, given* crs*, a statement* $\mathsf{st} \in \{0,1\}^n$ *and a proof* π*, either outputs* 1 *(accept) or* 0 *(reject).*

The following completeness and security properties should be satisfied:

- ***Completeness.*** *For any parameters* $\lambda, n \in \mathbb{N}$ *and any* $(\mathsf{st}, \mathsf{wit}) \in R$*, letting* $\mathsf{crs} = \mathsf{NIZK.Setup}(1^\lambda, 1^n)$*, and* $\pi = \mathsf{Prove}(\mathsf{crs}, (\mathsf{st}, \mathsf{wit}))$*, we have*

$$\mathsf{Verify}(\mathsf{crs}, \mathsf{st}, \pi) = 1$$

with probability 1*, where the probability is taken over the internal random coins used by the* $\mathsf{NIZK.Setup}$ *and* Prove *algorithms.*

- ***Computational Extractability.*** *There exists an efficient PPT extractor* $\mathsf{NIZK.Ext} = (\mathsf{NIZK.Ext}_1, \mathsf{NIZK.Ext}_2)$ *such that:*

  - *For any parameters* $\lambda, n \in \mathbb{N}$ *and for any polynomially bounded cheating prover* P^**, we have:*
  $$|\Pr[P^*(\mathsf{crs}) = 1] - \Pr[P^*(\widehat{\mathsf{crs}}) = 1]| \leq negl(\lambda),$$
  where $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^n)$ *and* $(\widehat{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Ext}_1(1^\lambda, 1^n)$.

  - *For any parameters* $\lambda, n \in \mathbb{N}$ *and for any polynomially bounded cheating prover* P^**, we have:*
  $$\Pr[(\mathsf{st}, \mathsf{wit}) \in R \quad \text{if} \quad \mathsf{Verify}(\widehat{\mathsf{crs}}, \mathsf{st}, \pi) = 1] = 1,$$
  where:

    1. $(\widehat{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Ext}_1(1^\lambda, 1^n)$,
    2. $(\mathsf{st}, \pi) \leftarrow P^*(\widehat{\mathsf{crs}})$*, and*
    3. $\mathsf{wit} = \mathsf{NIZK.Sim}_2((\widehat{\mathsf{crs}}, \tau), \mathsf{st}, \pi)$.

- ***Adaptive (Computational) Zero Knowledge.*** *There exists an efficient PPT simulator* $\mathsf{NIZK.Sim} = (\mathsf{NIZK.Sim}_1, \mathsf{NIZK.Sim}_2)$ *such that for any parameters* $\lambda, n \in \mathbb{N}$ *and for any non-uniform polynomially bounded "cheating" verifier* $V^* = (V_1^*, V_2^*)$*, we have*

$$|\Pr[V_2^*(\mathsf{crs}, \mathsf{st}, \pi, \xi) = 1 \wedge (\mathsf{st} \in L)] - \Pr[V_2^*(\widehat{\mathsf{crs}}, \mathsf{st}, \widehat{\pi}, \xi) = 1 \wedge (\mathsf{st} \in L)]| \leq negl(\lambda),$$

where:

  - ***Real Experiment:***
    1. $\mathsf{crs} = \mathsf{NIZK.Setup}(1^\lambda, 1^n)$
    2. $(\mathsf{st}, \mathsf{wit}, \xi) \leftarrow V_1^*(\mathsf{crs})$
    3. $\pi = \mathsf{Prove}(\mathsf{crs}, (\mathsf{st}, \mathsf{wit}))$

  - ***Ideal Experiment:***
    1. $(\widehat{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda, 1^n)$
    2. $(\mathsf{st}, \mathsf{wit}, \xi) \leftarrow V_1^*(\widehat{\mathsf{crs}})$
    3. $\widehat{\pi} = \mathsf{NIZK.Sim}_2((\widehat{\mathsf{crs}}, \tau), \mathsf{st})$.

A.2 Additively Homomorphic Encryption

Definition 13. ***(Additively Homomorphic Encryption).*** *An additively homomorphic encryption scheme* AHE *is a PKE scheme such that the message space* $\mathcal{M}$ *is associated with two efficiently computable operations* $(+, *)$ *and there exists the following PPT algorithms:*

- $\mathsf{AHE.Add}(\mathsf{ct}_0, \mathsf{ct}_1)$*: Takes as input two ciphertexts* ct_0 *and* ct_1 *and outputs a ciphertext* ct^**.*
- $\mathsf{AHE.ConstMul}(\mathsf{ct}_0, \mathsf{m}_1)$*: Takes as input a ciphertext* ct_0 *and a message* m_1 *and outputs a ciphertext* ct^**.*

such that the following properties are satisfied:

- ***Additive Homomorphism.*** *For any two* arbitrary *plaintext messages* m_0 *and* m_1*, letting* $(\mathsf{pk}, \mathsf{sk}) = \mathsf{PKE.Gen}(\lambda)$ *and* $\mathsf{ct}_b = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{m}_b)$ *for* $b \in \{0, 1\}$*, we have*

$$\mathsf{AHE.Dec}(\mathsf{sk}, \mathsf{AHE.Add}(\mathsf{ct}_0, \mathsf{ct}_1)) = \mathsf{m}_0 + \mathsf{m}_1.$$

- ***Constant Multiplication.*** *For any two* arbitrary *plaintext messages* m_0 *and* m_1*, letting* $(\mathsf{pk}, \mathsf{sk}) = \mathsf{PKE.Gen}(\lambda)$ *and* $\mathsf{ct}_0 = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{m}_0)$*, we have*

$$\mathsf{AHE.Dec}(\mathsf{sk}, \mathsf{AHE.ConstMul}(\mathsf{ct}_0, \mathsf{m}_1)) = \mathsf{m}_0 * \mathsf{m}_1.$$

A.3 Threshold Fully Homomorphic Encryption

Definition 14. ***(Threshold Fully Homomorphic Encryption (TFHE))*** *Let* $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *be a set of parties and let* $\mathbb{S}$ *be a class of efficient access structures on* $\mathcal{P}$*. A TFHE scheme for* $\mathbb{S}$ *is a tuple of PPT algorithms* $\mathsf{TFHE} = ($$\mathsf{TFHE.Gen}$, $\mathsf{TFHE.Enc}$, $\mathsf{TFHE.PartialDec}$, $\mathsf{TFHE.Eval}$, $\mathsf{TFHE.Combine}$$)$ *with the following properties:*

- $\mathsf{TFHE.Gen}(1^\lambda, 1^d, \mathbb{A})$*: On input the security parameter* λ*, a depth bound* d*, and an access structure* $\mathbb{A} \in \mathbb{S}$*, the setup algorithm outputs a public key* $\mathsf{pk}^{\mathsf{TFHE}}$*, and a set of secret key shares* $\mathsf{sk}_1^{\mathsf{TFHE}}, ..., \mathsf{sk}_n^{\mathsf{TFHE}}$*.*
- $\mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \mu)$*: On input a public key* $\mathsf{pk}^{\mathsf{TFHE}}$*, and a plaintext* μ*, the encryption algorithm outputs a ciphertext* ct*.*
- $\mathsf{TFHE.Eval}(\mathsf{pk}^{\mathsf{TFHE}}, \mathcal{C}, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$*: On input a public key* $\mathsf{pk}^{\mathsf{TFHE}}$*, a circuit* $\mathcal{C}$ *of depth at most* d*, and a set of ciphertexts* $\mathsf{ct}_1, \ldots, \mathsf{ct}_k$*, the evaluation algorithm outputs a cipehrtext* ct^**.*
- $\mathsf{TFHE.PartialDec}(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{ct})$*: On input a a secret key share* $\mathsf{sk}_i^{\mathsf{TFHE}}$ *and a ciphertext* ct*, the partial decryption algorithm outputs a partial decryption* μ_i*.*
- $\mathsf{TFHE.Combine}(\mathsf{pk}, \{\mu_i\}_{i \in \mathcal{S}})$*: On input a public key* pk *and a set of partial decryptions* $\{\mu_i\}_{i \in \mathcal{S}}$ *for some subset* $\mathcal{S} \subseteq \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$*, the combination algorithm either outputs a plaintext* μ *or the symbol* $\perp$*.*

As in a standard FHE scheme, we require that a TFHE scheme satisfies compactness, correctness, and security. We discuss these informally below. For formal definitions, refer [BGG+18].

Compactness. Informally, a TFHE scheme is said to be compact if the bit-length of a ciphertext output by the evaluation algorithm and the bit-length of any partial decryption of such a ciphertext is a priori upper bounded by some fixed polynomial, which is independent of the size of the circuit evaluated.

Correctness. Informally, a TFHE scheme is said to be correct if recombining partial decryptions of a ciphertext output by the evaluation algorithm returns the correct evaluation of the corresponding circuit on the underlying plaintexts.

Semantic Security. Informally, a TFHE scheme is said to provide semantic security if a PPT adversary cannot efficiently distinguish between encryptions of arbitrarily chosen plaintext messages μ_0 and μ_1, even given the secret key shares corresponding to a subset $\mathcal{S}$ of the parties, so long as $\mathcal{S}$ is an invalid access structure set.

Simulation Security: Informally, a TFHE scheme is said to provide simulation security if there exists an efficient algorithm $\mathsf{TFHE.Sim}$ that takes as input a circuit $\mathcal{C}$ of depth at most d, a set of ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_k$, and the output of $\mathcal{C}$ on the corresponding plaintexts, and outputs a set of partial decryptions corresponding to some subset of parties, such that its output is computationally indistinguishable from the output of a real algorithm that homomorphically evaluates the circuit $\mathcal{C}$ on the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_k$ and outputs partial decryptions using the corresponding secret key shares for the same subset of parties. In particular, the computational indistinguishability holds even when a PPT adversary is given the secret key shares corresponding to a subset $\mathcal{S}$ of the parties, so long as $\mathcal{S}$ is an invalid access structure set.

A.4 Secure Multiparty Computation

Parts of this section have been taken verbatim from [Gol04].

A multi-party protocol is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality. The security of a protocol is defined with respect to a functionality f. In particular, let n denote the number of parties. A non-reactive n-party functionality f is a (possibly randomized) mapping of n inputs to n outputs. A multiparty protocol with security parameter λ for computing a non-reactive functionality f is a protocol running in time $\mathsf{poly}(\lambda)$ and satisfying the following correctness requirement: if parties $P_1, \ldots, P_n$ with inputs $(x_1, \ldots, x_n)$ respectively, all run an honest execution of the protocol, then the joint distribution of the outputs $y_1, \ldots, y_n$ of the parties is statistically close to $f(x_1, \ldots, x_n)$.

A reactive functionality f is a sequence of non-reactive functionalities $f = (f_1, \ldots, f_\ell)$ computed in a stateful fashion in a series of phases. Let x_i^j denote the input of P_i in phase j, and let s^j denote the state of the computation after phase j. Computation of f proceeds by setting s^0 equal to the empty string and then computing $(y_1^j, \ldots, y_n^j, s^j) \leftarrow f_j(s^{j-1}, x_1^j, \ldots, x_n^j)$ for $j \in [\ell]$, where y_i^j denotes the output of P_i at the end of phase j. A multi-party protocol computing f also runs in ℓ phases, at the beginning of which each party holds an input and at the end of which each party obtains an output. (Note that parties may wait to decide on their phase-j input until the beginning of that phase.) Parties maintain state throughout the entire execution. The correctness requirement is that, in an honest execution of the protocol, the joint distribution of all the outputs $\{y_1^j, \ldots, y_n^j\}_{j=1}^{\ell}$ of all the phases is statistically close to the joint distribution of all the outputs of all the phases in a computation of f on the same inputs used by the parties.

Defining Security. We assume that readers are familiar with standard simulation-based definitions of secure multi-party computation in the standalone setting. We provide a self-contained definition for completeness and refer to [Gol04] for a more complete description. The security of a protocol (with respect to a functionality f) is defined by comparing the real-world execution of the protocol with an ideal-world evaluation of f by a trusted party. More concretely, it is required that for every adversary $\mathcal{A}$, which attacks the real execution of the protocol, there exist an adversary Sim, also referred to as a simulator, which can *achieve the same effect* in the ideal-world. Let's denote $\overrightarrow{x} = (x_1, \ldots, x_n)$.

The real execution In the real execution of the n-party protocol π for computing f is executed in the presence of an adversary Adv. The honest parties follow the instructions of π. The adversary Adv takes as input the security parameter k, the set $I \subset [n]$ of corrupted parties, the inputs of the corrupted parties, and an auxiliary input z. Adv sends all messages in place of corrupted parties and may follow an arbitrary polynomial-time strategy.

The interaction of Adv with a protocol π defines a random variable $\mathsf{REAL}_{\pi,\mathsf{Adv}(z),I}(k, \overrightarrow{x})$ whose value is determined by the coin tosses of the adversary and the honest players. This random variable contains the output of the adversary (which may be an arbitrary function of its view) as well as the outputs of the uncorrupted parties. We let $\mathsf{REAL}_{\pi,\mathsf{Adv}(z),I}$ denote the distribution ensemble $\{\mathsf{REAL}_{\pi,\mathsf{Adv}(z),I}(k, \overrightarrow{x})\}_{k \in \mathsf{N}, \langle \overrightarrow{x}, z \rangle \in \{0,1\}^*}$.

The ideal execution – security with abort . In this second variant of the ideal model, fairness and output delivery are no longer guaranteed. This is the standard relaxation used when a strict majority of honest parties is not assumed. In this case, an ideal execution for a function f proceeds as follows:

- **Send inputs to the trusted party:** As before, the parties send their inputs to the trusted party, and we let x'_i denote the value sent by P_i. Once again, for a semi-honest adversary we require $x'_i = x_i$ for all $i \in I$.
- **Trusted party sends output to the adversary:** The trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends $\{y_i\}_{i \in I}$ to the adversary.
- **Adversary instructs trust party to abort or continue:** This is formalized by having the adversary send either a continue or abort message to the trusted party. (A semi-honest adversary never aborts.) In the latter case, the trusted party sends to each uncorrupted party P_i its output value y_i. In the former case, the trusted party sends the special symbol $\perp$ to each uncorrupted party.
- **Outputs:** Sim outputs an arbitrary function of its view, and the honest parties output the values obtained from the trusted party.

The interaction of Sim with the trusted party defines a random variable $\mathsf{IDEAL}_{f_\perp,\mathsf{Adv}(z)}(k, \overrightarrow{x})$ as above,and we let $\{\mathsf{IDEAL}_{f_\perp,\mathsf{Adv}(z),I}(k, \overrightarrow{x})\}_{k \in \mathsf{N}, \langle \overrightarrow{x}, z \rangle \in \{0,1\}^*}$ where the subscript "$\perp$" indicates that the adversary can abort computation of f.

Having defined the real and the ideal worlds, we now proceed to define our notion of security.

Definition 15. *Let k be the security parameter. Let f be an n-party randomized functionality, and π be an n-party protocol for $n \in \mathsf{N}$.*

1. *We say that π t-securely computes f in the presence of malicious (resp., semi-honest) adversaries if for every PPT adversary (resp., semi-honest adversary)* Adv *there exists a PPT adversary (resp., semi-honest adversary)* Sim *such that for any $I \subset [n]$ with $|I| \leq t$ the following quantity is negligible:*

$$|Pr[\mathsf{REAL}_{\pi,\mathsf{Adv}(z),I}(k, \overrightarrow{x}) = 1] - Pr[\mathsf{IDEAL}_{f,\mathsf{Adv}(z),I}(k, \overrightarrow{x}) = 1]|$$

   where $\overrightarrow{x} = \{x_i\}_{i\in[n]} \in \{0,1\}^*$ and $z \in \{0,1\}^*$.

2. *Similarly, π t-securely computes f* with abort *in the presence of malicious adversaries if for every PPT adversary* Adv *there exists a super-polynomial time adversary* Sim *such that for any $I \subset [n]$ with $|I| \leq t$ the following quantity is negligible:*

$$|Pr[\mathsf{REAL}_{\pi,\mathsf{Adv}(z),I}(k, \overrightarrow{x}) = 1] - Pr[\mathsf{IDEAL}_{f_\perp,\mathsf{Adv}(z),I}(k, \overrightarrow{x}) = 1]|.$$

B Security Proofs

B.1 Proof of Theorem 1

Consider an environment $\mathcal{Z}$ who corrupts t^* parties where $t^* < t$. Additionally, for the two-round MPC protocol π used in our protocol π^{Any}, let $\pi.\mathsf{Ext}$ denote the extractor, that, on input the adversary's round one messages, extracts its inputs and let $\pi.\mathsf{Sim}.\mathsf{Setup}$ denote the algorithm used by $\pi.\mathsf{Sim}$ to compute the simulated CRS. The strategy of the simulator Sim for our protocol π^{Any} against the environment $\mathcal{Z}$ is described below. Let $\pi.\mathsf{Sim}$ denote the simulator of the underlying two round MPC. Further, let $\pi.\mathsf{Sim}$ use algorithm $(\pi.\mathsf{Sim}_1, \pi.\mathsf{Sim}_2)$ to compute the first and second round messages respectively. Note that since we consider a rushing adversary in the proof of security, the algorithm $\pi.\mathsf{Sim}_1(\cdot)$ does not require the adversary's input or output.

B.1.1 Description of Simulator

– **Setup:** Sim does the following:

  1. Generate $\mathsf{crs}_{\mathsf{sim}} \leftarrow \pi.\mathsf{Sim}.\mathsf{Setup}(1^\lambda)$.
  2. For each $i \in [n]$, compute $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$.
  3. For each $i, j \in [n]$, compute $(\mathsf{k}^{\mathsf{PRF}}_{i,j}, \mathsf{k}^{\mathsf{PRF}}_{j,i})$ as uniformly random strings.
  4. Query the ideal functionality with "Setup" and the list of corrupt parties. For each corrupt party $\mathcal{P}_i$, receive $(\mathsf{pp}^{\mathsf{TS}}, \mathsf{vk}^{\mathsf{TS}}, \mathsf{sk}^{\mathsf{TS}}_i)$ from the ideal functionality.
  5. For each $i \in [n]$, if $\mathcal{P}_i$ is corrupt, give $(\mathsf{crs}, \mathsf{pp}^{\mathsf{TS}}, \mathsf{vk}^{\mathsf{TS}}, \mathsf{sk}^{\mathsf{TS}}_i, \mathsf{sk}_i, \{\mathsf{vk}_j\}_{j\in[n]}, \{\mathsf{k}^{\mathsf{PRF}}_{j,i}, \mathsf{k}^{\mathsf{PRF}}_{i,j}\}_{j\in[n]})$ to the enivironment $\mathcal{Z}$.

– **Enrollment:** Sim does the following:

  – If it receives an enrollment query from the adversary on behalf of any corrupt party, it forwards it to the ideal functionality.
  – If an enrollment query was initiated by an honest party, Sim does nothing.

  – Then, for each enrollment query, for each corrupt party $\mathcal{P}_i$, Sim picks $\overrightarrow{\mathbf{w}}_i$ uniformly at random and forwards to $\mathcal{Z}$.

SignOn Phase: Case 1 - Honest Party as $\mathcal{P}^*$
Suppose an honest party $\mathcal{P}^*$ uses an input vector u and a message msg for which it wants a token by interacting with a set of parties S. Sim gets the tuple (msg, S) from the ideal functionality $\mathcal{F}_{\mathsf{DiFuz}}$ and interacts with the adversary $\mathcal{A}$ as below:

– **Round 1:** $(\mathsf{Sim} \rightarrow)$ [9] Sim sends (msg, S) to the adversary $\mathcal{A}$ for each corrupt party $\mathcal{P}_i \in S$.

– **Round 2:** $(\rightarrow \mathsf{Sim})$ On behalf of each corrupt party $\mathcal{P}_i \in S$, receive $(\mathsf{msg}_{1,i}, \sigma_{1,i})$ from the adversary.

– **Round 3:** $(\mathsf{Sim} \rightarrow)$ Sim does the following:

  1. On behalf of each honest party $\mathcal{P}_j$ in $S \setminus \mathcal{P}^*$, compute $\mathsf{msg}_{1,j} \leftarrow \pi.\mathsf{Sim}_1(1^\lambda, \mathcal{P}_j)$ and $\sigma_{1,j} = \mathsf{Sign}(\mathsf{sk}_j, \mathsf{msg}_{1,j})$.
  2. Let $\mathsf{Trans}_{\mathsf{DiFuz}}$ denote the set of tuples of the form $(\mathsf{msg}_{1,i}, \sigma_{1,i})$ received in round 2 and computed in the above step.
  3. Compute the simulated first round message of protocol π on behalf of honest party $\mathcal{P}^*$ as follows: $\mathsf{msg}_{1,*} \leftarrow \pi.\mathsf{Sim}_1(1^\lambda, \mathcal{P}^*)$.
  4. Send $(\mathsf{Trans}_{\mathsf{DiFuz}}, \mathsf{msg}_{1,*})$ to the adversary for each corrupt party $\mathcal{P}_i \in S$.

– **Round 4:** $(\rightarrow \mathsf{Sim})$ For each corrupt party $\mathcal{P}_i \in S$, receive $(\mathsf{ct}_i, \{\mathsf{ek}_{j,i}\}_{j \in S})$ from the adversary.

– **Message to Ideal Functionality $\mathcal{F}_{\mathsf{DiFuz}}$:** Sim does the following:

  1. Run $\pi.\mathsf{Sim}(\cdot)$ on the transcript of the underlying protocol π.
  2. If $\pi.\mathsf{Sim}(\cdot)$ decides to instruct the ideal functionality of π to deliver output to the honest party $\mathcal{P}^*$ in protocol π, then so does Sim to the functionality $\mathcal{F}_{\mathsf{DiFuz}}$ in our distributed fuzzy secure authentication protocol. Note that in order to do so, $\pi.\mathsf{Sim}(\cdot)$ might internally use the algorithm $\pi.\mathsf{Ext}(\cdot)$. Essentially, this step guarantees Sim that the adversary behaved honestly in the protocol.
  3. Else, Sim outputs $\perp$.

SignOn Phase: Case 2 - Malicious Party as $\mathcal{P}^*$
Suppose a malicious party is the initiator $\mathcal{P}^*$. Sim interacts with the adversary $\mathcal{A}$ as below:

– **Round 1:** $(\rightarrow \mathsf{Sim})$ Sim receives (msg, S) from the adversary $\mathcal{A}$ on behalf of each honest party $\mathcal{P}_j$.

– **Round 2:** $(\mathsf{Sim} \rightarrow)$ Sim does the following:

  1. On behalf of each honest party $\mathcal{P}_j$ in S, compute and send the pair $\mathsf{msg}_{1,j} \leftarrow \pi.\mathsf{Sim}_1(1^\lambda, \mathcal{P}_j)$ and $\sigma_{1,j} = \mathsf{Sign}(\mathsf{sk}_j, \mathsf{msg}_{1,j})$ to the adversary.

[9]The arrowhead denotes that in this round messages are outgoing from the simulator.

– **Round 3:** ($\rightarrow \mathsf{Sim}$) Sim receives a tuple $(\mathsf{Trans}_{\mathsf{DiFuz}}, \mathsf{msg}_{1,*})$ from the adversary $\mathcal{A}$ on behalf of each honest party $\mathcal{P}_j$.

– **Round 4:** ($\mathsf{Sim} \rightarrow$) Sim does the following:

1. On behalf of each honest party $\mathcal{P}_j$, do the following:
   1. Let $\mathsf{Trans}_{\mathsf{DiFuz}}$ consist of a set of messages of the form $(\mathsf{msg}_{1,i}, \sigma_{1,i})$, $\forall i \in S \setminus \mathcal{P}^*$. Output $\perp$ if $\mathsf{Verify}(\mathsf{vk}_i, \mathsf{msg}_{1,i}, \sigma_{1,i}) \neq 1$.
   2. Let τ_1 denote the transcript of protocol π after round 1. That is, $\tau_1 = \{\mathsf{msg}_{1,i}\}_{i \in S}$.
2. Let τ_2 denote the subset of τ_1 corresponding to all the messages generated by honest parties.
3. If τ_2 not equal for all the honest parties, output "$\mathsf{SpecialAbort}$".
4. If $\mathsf{msg}_{1,*}$ not equal for all the honest parties, set a variable $\mathsf{flag} = 0$.
5. **Query to Ideal Functionality $\mathcal{F}_{\mathsf{DiFuz}}$:**
   1. Compute $\mathsf{inp}_{\mathcal{A}} = \pi.\mathsf{Ext}(\tau_1, \mathsf{crs}_{\mathsf{sim}})$.
   2. Query the ideal functionality $\mathcal{F}_{\mathsf{DiFuz}}$ with $\mathsf{inp}_{\mathcal{A}}$ to receive output $\mathsf{out}_{\mathcal{A}}$.
6. Compute the set of second round messages $\mathsf{msg}_{2,j}$ of protocol π on behalf of each honest party $\mathcal{P}_j$ as $\pi.\mathsf{Sim}_2(\tau_1, \mathsf{inp}_{\mathcal{A}}, \mathsf{out}_{\mathcal{A}}, \mathcal{P}_j)$.
7. On behalf of each honest party $\mathcal{P}_j$, do the following:
   1. Let $(\mathsf{Trans}_{\mathsf{DiFuz}}, \mathsf{msg}_{1,*})$ denote the message received from the adversary in round 3. Compute $\mathsf{ek}_j = \oplus_{i \in S} \mathsf{PRF}(\mathsf{k}^{\mathsf{PRF}}_{j,i}, \mathsf{msg}_{1,*})$.
   2. If $\mathsf{flag} = 0$, compute $\mathsf{ct}_j = \mathsf{SKE.Enc}(\mathsf{rand}, 0^{|\mathsf{msg}_{2,j}|})$ where rand is a string chosen uniformly at random.
   3. Else, compute $\mathsf{ct}_j = \mathsf{SKE.Enc}(\mathsf{ek}_j, \mathsf{msg}_{2,j})$.
   4. For each party $\mathcal{P}_i \in S$, compute $\mathsf{ek}_{i,j} = \mathsf{PRF}(\mathsf{k}^{\mathsf{PRF}}_{i,j}, \mathsf{msg}_{1,*})$.
   5. Send $(\mathsf{ct}_j, \{\mathsf{ek}_{i,j}\}_{i \in S})$ to the adversary.

B.1.2 Hybrids

We now show that the above simulation strategy is successful against all environments $\mathcal{Z}$. That is, the view of the corrupt parties along with the output of the honest parties is computationally indistinguishable in the real and ideal worlds. We will show this 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 parties as in the real world.

2. Hyb_1 **- Special Abort:** In this hybrid, SimHyb outputs "$\mathsf{SpecialAbort}$" as done by Sim in round 4 of Case 2 of the simulation strategy. That is, SimHyb outputs "$\mathsf{SpecialAbort}$" if all the signatures verify but the adversary does not send the same transcript of the first round of protocol π to all the honest parties.

3. Hyb_2 **- Simulate MPC messages:** In this hybrid, SimHyb does the following:

   – In the setup phase, compute the CRS as $\mathsf{crs}_{\mathsf{sim}} \leftarrow \pi.\mathsf{Sim.Setup}(1^{\lambda})$.

- **Case 1:** Suppose an honest party plays the role of $\mathcal{P}^*$, do:
  - In round 3, compute the first round messages $\mathsf{msg}_{1,j}$ of protocol π on behalf of every honest party $\mathcal{P}_j \in S$ and the first round message $\mathsf{msg}_{1,*}$ on behalf of the party $\mathcal{P}^*$ by running the algorithm $\pi.\mathsf{Sim}_1(\cdot)$ as done in the ideal world.
  - Then, instead of $\mathcal{P}^*$ computing the output by itself using the protocol messages, instruct the ideal functionality to deliver output to $\mathcal{P}^*$. That is, execute the "message to ideal functionality" step exactly as in the ideal world.
- **Case 2:** Suppose a corrupt party plays the role of $\mathcal{P}^*$, do:
  - In round 2, compute the first round messages $\mathsf{msg}_{1,j}$ of protocol π on behalf of every honest party $\mathcal{P}_j \in S$ by running the algorithm $\pi.\mathsf{Sim}_1(\cdot)$ as done in the ideal world.
  - Interact with the ideal functionality exactly as done by Sim in the ideal world. That is, query the ideal functionality on the output of the extractor $\pi.\mathsf{Ext}(\cdot)$ on input $(\tau_1, \mathsf{crs}_{\mathsf{sim}})$ and receive output $\mathsf{out}_{\mathcal{A}}$.
  - Compute the set of second round messages $\mathsf{msg}_{2,j}$ of protocol π on behalf of each honest party $\mathcal{P}_j$ as $\pi.\mathsf{Sim}_2(\tau_1, \mathsf{inp}_{\mathcal{A}}, \mathsf{out}_{\mathcal{A}}, \mathcal{P}_j)$.

4. Hyb_3 **- Switch PRF Output in Case 2:** In this hybrid, suppose a corrupt party plays the role of $\mathcal{P}^*$, SimHyb computes the value of the variable flag as done by the simulator Sim in round 4 of the simulation strategy. That is, SimHyb sets $\mathsf{flag} = 0$ if the adversary did not send the same round 1 messages of protocol π to all the honest parties $\mathcal{P}_j \in S$. Then, on behalf of every honest party $\mathcal{P}_j$, SimHyb does the following:
   - If $\mathsf{flag} = 1$, compute ct_j as in Hyb_1.
   - If $\mathsf{flag} = 0$, compute $\mathsf{ct}_j = \mathsf{SKE.Enc}(\mathsf{rand}, \mathsf{msg}_{2,j})$ where rand is chosen uniformly at random and not as the output of the PRF anymore.

5. Hyb_4 **- Switch Ciphertext in Case 2:** In this hybrid, suppose a corrupt party plays the role of $\mathcal{P}^*$, SimHyb does the following: if $\mathsf{flag} = 0$, compute $\mathsf{ct}_j = \mathsf{SKE.Enc}(\mathsf{rand}, 0^{|\mathsf{msg}_{2,j}|})$ as in the ideal world.

6. Hyb_5 **- Switch Template Shares:** In this hybrid, for each corrupt party $\mathcal{P}_i$, for each enrollment query, SimHyb picks the vector $\vec{\mathbf{u}}_i$ as a uniformly random vector as in the ideal world instead of as a secret sharing of the template $\vec{\mathbf{u}}$. This hybrid corresponds to the ideal world.

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

Lemma 1. *Assuming the strong unforgeability of the signature scheme,* Hyb_0 *is computationally indistinguishable from* Hyb_1*.*

Proof. The only difference between the two hybrids is that in Hyb_1, SimHyb might output "$\mathsf{SpecialAbort}$". We now show that SimHyb outputs "$\mathsf{SpecialAbort}$" in Hyb_1 only with negligible probability.

Suppose not. That is, suppose there exists an environment $\mathcal{Z}$ that can cause SimHyb to output "$\mathsf{SpecialAbort}$" in Hyb_1 with non-negligible probability, then we will use $\mathcal{Z}$ to construct an adversary $\mathcal{A}_{\mathsf{Sign}}$ that breaks the strong unforgeability of the signature scheme which is a contradiction.

$\mathcal{A}_\pi$ begins an execution of the DiFuz protocol interacting with the environment $\mathcal{Z}$ as in Hyb_1. For each honest party $\mathcal{P}_j$, $\mathcal{A}_{\mathsf{Sign}}$ interacts with a challenger $\mathcal{C}_{\mathsf{Sign}}$ and gets a verification key vk_j which is forwarded to $\mathcal{Z}$ as part of the setup phase of DiFuz. Then, during the course of the protocol, $\mathcal{A}_{\mathsf{Sign}}$ forwards signature queries from $\mathcal{Z}$ to $\mathcal{C}_{\mathsf{Sign}}$ and the responses from $\mathcal{C}_{\mathsf{Sign}}$ to $\mathcal{Z}$.

Finally, suppose $\mathcal{Z}$ causes $\mathcal{A}_{\mathsf{Sign}}$ to output "$\mathsf{SpecialAbort}$" with non-negligible probability. Then, it must be the case that for some tuple of the form $(\mathsf{msg}_{1,j}, \sigma_{1,j})$ corresponding to an honest party $\mathcal{P}_j$, the signature $\sigma_{1,j}$ was not forwarded to $\mathcal{Z}$ from $\mathcal{C}_{\mathsf{Sign}}$ but still verified successfully. Thus, $\mathcal{A}_{\mathsf{Sign}}$ can output the same tuple $(\mathsf{msg}_{1,j}, \sigma_{1,j})$ as a forgery to break the strong unforgeability of the signature scheme with non-negligible probability which is a contradiction. □

Lemma 2. *Assuming the security of the MPC protocol π, Hyb_1 is computationally indistinguishable from Hyb_2.*

Proof. Suppose there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability. We will use $\mathcal{Z}$ to construct an adversary $\mathcal{A}_\pi$ that breaks the security of the protocol π which is a contradiction.

$\mathcal{A}_\pi$ begins an execution of the DiFuz protocol interacting with the environment $\mathcal{Z}$ and an execution of protocol π for evaluating circuit $\mathcal{C}$ (Figure 2) interacting with a challenger $\mathcal{C}_\pi$. Now, suppose $\mathcal{Z}$ corrupts a set of parties $\mathcal{P}$, $\mathcal{A}_\pi$ corrupts the same set of parties in the protocol π. First, the registration phase of protocol DiFuz takes place. Then, $\mathcal{A}_\pi$ receives a string crs from the challenger $\mathcal{C}_\pi$ which is either honestly generated or simulated. $\mathcal{A}_\pi$ sets this string to be the crs in the setup phase of the DiFuz protocol with $\mathcal{A}$. The rest of the setup protocol is run exactly as in Hyb_0.

Case 1: Honest party as $\mathcal{P}^*$
Now, since we consider a rushing adversary for protocol π, on behalf of every honest party $\mathcal{P}_j$, $\mathcal{A}_\pi$ first receives a message msg_j from the challenger $\mathcal{C}_\pi$. $\mathcal{A}_\pi$ sets msg_j to be the message $\mathsf{msg}_{1,j}$ in round 3 of its interaction with $\mathcal{Z}$ and then computes the rest of its messages to be sent to $\mathcal{Z}$ exactly as in Hyb_1. $\mathcal{A}_\pi$ receives a set of messages corresponding to protocol π from $\mathcal{Z}$ on behalf of the corrupt parties in $\mathcal{P}$ which it forwards to $\mathcal{C}_\pi$ as its own messages for protocol π.

Case 2: Corrupt party as $\mathcal{P}^*$
As in the previous case, on behalf of every honest party $\mathcal{P}_j$, $\mathcal{A}_\pi$ first receives a message msg_j from the challenger $\mathcal{C}_\pi$. $\mathcal{A}_\pi$ sets msg_j to be the message $\mathsf{msg}_{1,j}$ in round 2 of its interaction with $\mathcal{Z}$. Then, in round 4, if the signatures verify, $\mathcal{A}_\pi$ forwards the set of messages corresponding to protocol π received from $\mathcal{Z}$ on behalf of the corrupt parties in $\mathcal{P}$ to $\mathcal{C}_\pi$ as its own messages for protocol π. Then, on behalf of every honest party $\mathcal{P}_j$, $\mathcal{A}_\pi$ receives a message msg_j from the challenger $\mathcal{C}_\pi$ as the second round message of protocol π. $\mathcal{A}_\pi$ sets msg_j to be the message $\mathsf{msg}_{2,j}$ in round 4 of its interaction with $\mathcal{Z}$ and computes the rest of its messages to be sent to $\mathcal{Z}$ exactly as in Hyb_1.

Notice that when the challenger $\mathcal{C}_\pi$ sends honestly generated messages, the experiment between $\mathcal{A}_\pi$ and $\mathcal{Z}$ corresponds exactly to Hyb_1 and when the challenger $\mathcal{C}_\pi$ sends simulated messages, the experiment corresponds exactly to Hyb_2. Thus, if $\mathcal{Z}$ can distinguish between the two hybrids with non-negligible probability, $\mathcal{A}_\pi$ can use the same guess to break the security of the scheme π with non-negligible probability which is a contradiction. $\square$

Lemma 3. *Assuming the security of the pseudorandom function,* Hyb_2 *is computationally indistinguishable from* Hyb_3*.*

Proof. Suppose there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability. We will use $\mathcal{Z}$ to construct an adversary $\mathcal{A}_{\mathsf{PRF}}$ that breaks the security of the pseudorandom function which is a contradiction.

The adversary $\mathcal{A}_{\mathsf{PRF}}$ interacts with the environment $\mathcal{Z}$ in an execution of the protocol DiFuz. For each honest party $\mathcal{P}_j$, $\mathcal{A}_{\mathsf{PRF}}$ also interacts with a challenger $\mathcal{C}_{\mathsf{PRF}}$ in the PRF security game. For each j, $\mathcal{C}_{\mathsf{PRF}}$ sends the PRF keys corresponding to the set of corrupt parties ($< k$) as requested by $\mathcal{A}_{\mathsf{PRF}}$ which is then forwarded to $\mathcal{Z}$ during the setup phase. Then, $\mathcal{A}_{\mathsf{PRF}}$ continues interacting with $\mathcal{Z}$ up to round 3 as in Hyb_1. Now, in round 4, suppose it computes the value of the variable flag to be 0 (as computed in Hyb_3), then $\mathcal{A}_{\mathsf{PRF}}$ does the following: for each honest party $\mathcal{P}_j$, forward the message $\mathsf{msg}_{1,*}$ received in round 3. Then, set the XOR of the set of responses from $\mathcal{C}_{\mathsf{PRF}}$ to be the value ek_j used for generating the ciphertext ct_j.

Now notice that when the challenger $\mathcal{C}_{\mathsf{PRF}}$ responds with a set of honest PRF evaluations for each honest party j, the interaction between $\mathcal{A}_{\mathsf{PRF}}$ and $\mathcal{Z}$ exactly corresponds to Hyb_2 and when the challenger responds with a set of uniformly random strings, the interaction between $\mathcal{A}_{\mathsf{PRF}}$ and $\mathcal{Z}$ exactly corresponds to Hyb_3. Thus, if $\mathcal{Z}$ can distinguish between the two hybrids with non-negligible probability, $\mathcal{A}_{\mathsf{DPRF}}$ can use the same guess to break the pseudorandomness property of the PRF scheme with non-negligible probability which is a contradiction. $\square$

Lemma 4. *Assuming the semantic security of the secret key encryption scheme,* Hyb_3 *is computationally indistinguishable from* Hyb_4*.*

Proof. Suppose there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability. We will use $\mathcal{Z}$ to construct an adversary $\mathcal{A}_{\mathsf{SKE}}$ that breaks the semantic security of the encryption scheme which is a contradiction.

The adversary $\mathcal{A}_{\mathsf{SKE}}$ interacts with the environment $\mathcal{Z}$ as in Hyb_3. Then, on behalf of every honest party $\mathcal{P}_j$, before sending its round 4 message, $\mathcal{A}_{\mathsf{SKE}}$ first sends the tuple $(\mathsf{msg}_{2,j}, 0^{|\mathsf{msg}_{2,j}|})$ to the challenger $\mathcal{C}_{\mathsf{SKE}}$ of the secret key encryption scheme. Corresponding to every honest party, it receives a ciphertext which is either an encryption of $\mathsf{msg}_{2,j}$ or $0^{|\mathsf{msg}_{2,j}|}$ using a secret key chosen uniformly at random. Then, $\mathcal{A}_{\mathsf{SKE}}$ sets this ciphertext to be the value ct_j and continues interacting with the environment $\mathcal{Z}$ exactly as in Hyb_2. Notice that when the challenger $\mathcal{C}_{\mathsf{SKE}}$ sends ciphertexts of $\mathsf{msg}_{2,j}$, the experiment between $\mathcal{A}_{\mathsf{SKE}}$ and $\mathcal{Z}$ corresponds exactly to Hyb_3 and when the challenger $\mathcal{C}_{\mathsf{SKE}}$ sends ciphertexts of $0^{|\mathsf{msg}_{2,j}|}$, the experiment corresponds exactly to Hyb_4. Thus, if $\mathcal{Z}$ can distinguish between the two hybrids with non-negligible probability, $\mathcal{A}_{\mathsf{SKE}}$ can use the same guess to break the semantic security of the encryption scheme with non-negligible probability which is a contradiction. $\square$

Lemma 5. *Assuming the security of the secret sharing scheme,* Hyb_4 *is computationally indistinguishable from* Hyb_5*.*

Proof. The only difference between the two hybrids is that in Hyb_4, for every enrollment with a template $\overrightarrow{\mathbf{u}}$, each corrupt party $\mathcal{P}_i$ gets a vector $\overrightarrow{\mathbf{u}}_i$ that is picked as a share of $\overrightarrow{\mathbf{u}}$ by running the secret sharing algorithm Share whereas, in Hyb_5, these values are picked uniformly at random by the simulator. Since the number of corrupt parties is less than the threshold used in the secret sharing scheme and no information about any other honest party's share is known to the simulator, it is easy to see that if there exists an environment that can distinguish between these two hybrids, we can build an adversary $\mathcal{A}$ that breaks the security of the threshold secret sharing scheme which is a contradiction.

$\square$

B.2 Proof of Theorem 3

Consider an environment $\mathcal{Z}$ who corrupts t^* parties where $t^* < t$. Additionally, let $\mathsf{NIZK.Sim}$ and $\mathsf{NIZK.Ext}$ denote the simulator and extractor algorithms for the NIZK argument system used in our protocol $\pi^{\mathsf{Any-TFHE}}$. The strategy of the simulator Sim for our protocol against the environment $\mathcal{Z}$ is described below. Note that throughout the description, we assume that Sim maintains a record/transcript corresponding to each completed session, and that it can efficiently retrieve the record corresponding to a past session at any point of time.

B.2.1 Description of Simulator

Setup Phase:

In the setup phase, the simulator Sim executes the following algorithm:

1. Generate $(\mathsf{pk}^{\mathsf{TFHE}}, \mathsf{sk}_1^{\mathsf{TFHE}}, \ldots, \mathsf{sk}_n^{\mathsf{TFHE}}) \leftarrow \mathsf{TFHE.Gen}(1^\lambda, n, t)$.
2. For each $i \in [n]$, compute $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$.
3. Interact with the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ as follows:
    1. Issue a "`Setup`" query to $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$.
    2. Receive in response a tuple of the form ("`KeyShares`", $\{\mathsf{sk}_i^{\mathsf{TS}}\}_{i \in C}$), where $C \subseteq [n]$ contains the identities of all corrupt parties.
4. For each corrupt party $\mathcal{P}_i$, compute $\mathsf{com}_i = \mathsf{Commit}(\mathsf{sk}_i^{\mathsf{TFHE}}; \mathsf{r}_i^{\mathsf{com}})$.
5. For each honest party $\mathcal{P}_i$, compute $\mathsf{com}_i = \mathsf{Commit}(0^\lambda; \mathsf{r}_i^{\mathsf{com}})$.
6. Provide the following to each corrupt party $\mathcal{P}_i$ for $i \in C$:

$$(\mathsf{pk}^{\mathsf{TFHE}}, \mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{pp}^{\mathsf{TS}}, \mathsf{vk}^{\mathsf{TS}}, \mathsf{sk}_i^{\mathsf{TS}}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_n), \mathsf{sk}_i, (\mathsf{com}_1, \ldots, \mathsf{com}_n), \mathsf{r}_i^{\mathsf{com}}).$$

Registration Phase

Suppose that some party $\mathcal{P}^*$ issues a query of the form ("`Register`", sid, $\overrightarrow{\mathbf{w}}$) to the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$. If $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ responds with ("`Registered`", $\mathsf{sid}, \mathcal{P}_i$), the simulator Sim executes the following algorithm:

1. Create the following dummy ciphertext: $\mathsf{ct}_{\overrightarrow{\mathbf{w}}} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \overrightarrow{\mathbf{w}}^*)$, where $\overrightarrow{\mathbf{w}}^*$ is any arbitrarily chosen template independent of the actual template $\overrightarrow{\mathbf{w}}$.
2. Send $\mathsf{ct}_{\overrightarrow{\mathbf{w}}}$ to each corrupt party $\mathcal{P}_i$ for $i \in C$.

SignOn Phase: Case 1 - Honest Party as $\mathcal{P}^*$

Suppose that in some session with id sid, an honest party $\mathcal{P}^*$ issues a query of the form ("$\texttt{SignOn}$", sid, vk, msg, $\mathcal{P}^*$, $\overrightarrow{\mathbf{u}}$, $S \subseteq [n]$), where $\overrightarrow{\mathbf{u}}$ is the measurement, msg is the message for which $\mathcal{P}^*$ wants a token, and $S \subseteq [n]$ is the set of t parties with which $\mathcal{P}^*$ wishes to interact.

Let $S = S_0 \cup S_1$, where S_0 is the set of all honest parties in S and S_1 is the set of all corrupt parties in S. Sim interacts with the ideal functionality $\mathcal{F}^{\mathsf{TS}}_{\text{FTT}}$ and the adversary $\mathcal{A}$ as follows.

– **Initial Step:** Sim receives the message ("$\texttt{Signing Req}$", sid, msg, $\mathcal{P}_i$) issued by the ideal functionality $\mathcal{F}^{\mathsf{TS}}_{\text{FTT}}$ to each corrupt party $\mathcal{P}_i$ in the set S.

– **Round 1:** ($\mathsf{Sim} \rightarrow$) On behalf of party $\mathcal{P}^*$, Sim does the following:

1. Compute a dummy ciphertext $\mathsf{ct}_{\overrightarrow{\mathbf{u}}} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \overrightarrow{\mathbf{u}}^*)$, where $\overrightarrow{\mathbf{u}}^*$ is any arbitrarily chosen measurement.
2. Compute a simulated proof $\pi_{\overrightarrow{\mathbf{u}}} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{\overrightarrow{\mathbf{u}}})$ for $\mathsf{st}_{\overrightarrow{\mathbf{u}}} = (\mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{pk}^{\mathsf{TFHE}}) \in L_1$.
3. To each corrupt party $\mathcal{P}_i \in S_1$, send $(\mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \pi_{\overrightarrow{\mathbf{u}}})$.

– **Round 2:** ($\rightarrow \mathsf{Sim}$) On behalf of party $\mathcal{P}^*$, Sim receives a tuple $(\mathsf{ct}_{K_i}, \pi_{K_i}, \sigma_{i,0}, \sigma_{i,1})$ from each corrupt party $\mathcal{P}_i \in S_1$.

– **Round 3:** ($\mathsf{Sim} \rightarrow$) Sim proceeds as follows:

1. **TFHE Ciphertext Validation-1:** In order to check that each corrupt party $\mathcal{P}_i \in S_1$ generated ct_{K_i} honestly, Sim does the following:
   1. Extract the one-time key K_i as:
   $$(K_i, \mathsf{r}_{K_i}) = \mathsf{NIZK.Ext}(\pi_{K_i}, \mathsf{st}_{K_i}),$$
   where $\mathsf{st}_{K_i} = (\mathsf{ct}_{K_i}, \mathsf{pk}^{\mathsf{TFHE}})$.
   2. If $\mathsf{ct}_{K_i} \neq \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, K_i; \mathsf{r}_{K_i})$, output $\perp$ and abort.
2. Otherwise, on behalf of each honest party $\mathcal{P}_j$ in the set S_0, Sim does the following:
   1. Compute a dummy ciphertext $\mathsf{ct}_{K_j} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, 0^\lambda)$.
   2. Compute a simulated proof $\pi_{K_j} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{K_j})$ for the statement $\mathsf{st}_{K_j} = (\mathsf{ct}_{K_j}, \mathsf{pk}^{\mathsf{TFHE}}) \in L_1$.
   3. Compute the signatures as:
   $$\sigma_{j,0} = \mathsf{Sign}(\mathsf{sk}_j, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}), \quad \sigma_{j,1} = \mathsf{Sign}(\mathsf{sk}_j, \mathsf{ct}_{K_j}).$$
3. Finally, on behalf of $\mathcal{P}^*$, Sim forwards the following to each corrupt party in S_1:
$$\{(\mathsf{ct}_{K_\ell}, \pi_{K_\ell}, \sigma_{\ell,0}, \sigma_{\ell,1})\}_{\mathcal{P}_\ell \in S}.$$

– **Round 4:** ($\rightarrow \mathsf{Sim}$) On behalf of party $\mathcal{P}^*$, Sim receives the following from each corrupt party $\mathcal{P}_i \in S_1$:
$$(\mathsf{ct}_i, \{(\pi_{i,\ell}, \mu_{i,\ell})\}_{\mathcal{P}_\ell \in S}).$$

– **TFHE Ciphertext Validation-2:** At this point, Sim performs the following validation step: in order to check that each corrupt party $\mathcal{P}_i \in S_1$ generated $\mu_{i,\ell}$ honestly for each party $\mathcal{P}_\ell \in S$, Sim does the following:

1. Extract the TFHE partial decryption key $\mathsf{sk}_i^{\mathsf{TFHE}}$ as:

$$(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{r}_i^{\mathsf{com}}) - \mathsf{NIZK.Ext}(\pi_{i,\ell}, \mathsf{st}_{i,\ell}),$$

where $\mathsf{st}_{i,\ell} = (\mathsf{ct}_{\mathcal{C},\ell}, \mu_{i,\ell}, \mathsf{com}_i)$ and $\mathcal{P}_\ell$ is some party in the set S.

2. If $\mathsf{com}_i \neq \mathsf{Commit}(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{r}_i^{\mathsf{com}})$ or $\mu_{i,\ell} \neq \mathsf{TFHE.PartialDec}(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{ct}_{\mathcal{C},\ell})$ for some party $\mathcal{P}_\ell \in S$, output $\perp$ and abort.

– **Signature Validation:** Next, Sim performs the following validation steps:

1. In order to check that each corrupt party $\mathcal{P}_i \in S_1$ generated $\sigma_{i,0}$ and $\sigma_{i,1}$ honestly, Sim checks if $\mathsf{Verify}(\mathsf{vk}_i, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \sigma_{i,0}) = 1$ and $\mathsf{Verify}(\mathsf{vk}_i, \mathsf{ct}_{K_i}, \sigma_{i,1}) = 1$. If not, it outputs $\perp$ and aborts.

– **Token Validation:** In order to check that each corrupt party $\mathcal{P}_i \in S_1$ generated ct_i honestly, Sim does the following for each corrupt party $\mathcal{P}_i \in S_1$:

1. Use the previously extracted one-time key K_i to recover the corresponding partial token on the message msg as: $\mathsf{Token}_i = K_i \oplus \mathsf{ct}_i z$.
2. If $\mathsf{Token}_i \neq \mathsf{TS.Sign}(\mathsf{sk}_i^{\mathsf{TS}}, \mathsf{msg})$, output $\perp$ and abort.

– **Message to Ideal Functionality:** Send (“`Agreed`”$, \mathsf{sid}, \mathsf{msg}, \mathcal{P}_i$) to the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ on behalf of each corrupt party $\mathcal{P}_i \in S_1$. In other words, Sim instructs the ideal functionality to deliver the output to $\mathcal{P}^*$ depending on whether the measurement matches the template.

SignOn Phase: Case 2 - Malicious Party as $\mathcal{P}^*$

Suppose that in some session with id sid, a corrupt party $\mathcal{P}^*$ issues a query of the form (“`SignOn`”, sid, vk, msg, $\mathcal{P}^*$, $\overrightarrow{\mathbf{u}}$, $S \subseteq [n]$), where $\overrightarrow{\mathbf{u}}$ is the measurement, msg is the message for which $\mathcal{P}^*$ wants a token, and $S \subseteq [n]$ is the set of $\mathtt{t}$ parties with which $\mathcal{P}^*$ wishes to interact.

As before, let $S = S_0 \cup S_1$, where S_0 is the set of all honest parties in S and S_1 is the set of all corrupt parties in S. Sim initializes a variable flag to 0, and interacts with the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ and the adversary $\mathcal{A}$ as follows.

– **Round 1:** ($\rightarrow \mathsf{Sim}$) On behalf of each honest party $\mathcal{P}_j \in S_0$, Sim does the following:

1. Receive a ciphertext $\mathsf{ct}_{\overrightarrow{\mathbf{u}},j}$ and a proof $\pi_{\overrightarrow{\mathbf{u}},j}$ from the party $\mathcal{P}^*$.
2. Extract the corresponding measurement $\overrightarrow{\mathbf{u}}$ as:

$$(\overrightarrow{\mathbf{u}}, \mathsf{r}_{\overrightarrow{\mathbf{u}},j}) = \mathsf{NIZK.Ext}(\pi_{\overrightarrow{\mathbf{u}},j}, \mathsf{st}_{\overrightarrow{\mathbf{u}},j}),$$

where $\mathsf{st}_{\overrightarrow{\mathbf{u}},j} = (\mathsf{ct}_{\overrightarrow{\mathbf{u}},j}, \mathsf{pk}^{\mathsf{TFHE}})$.

3. Output $\perp$ and abort if $\mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \overrightarrow{\mathbf{u}}; \mathsf{r}_{\overrightarrow{\mathbf{u}},j}) \neq \mathsf{ct}_{\overrightarrow{\mathbf{u}},j}$.

- **Routing-Consistency Check-1:** Sim checks if there exists a pair of honest parties $(\mathcal{P}_j, \mathcal{P}_{j'})$ such that $\mathsf{ct}_{\vec{\mathbf{u}},j} \neq \mathsf{ct}_{\vec{\mathbf{u}},j'}$:
  1. If such a pair of honest parties exists, Sim sets flag to 1.
  2. Else, Sim sets $\mathsf{ct}_{\vec{\mathbf{u}}} = \mathsf{ct}_{\vec{\mathbf{u}},j}$ for any honest party $\mathcal{P}_j \in S_0$.

- **Round 2: ($\mathsf{Sim} \rightarrow$)** On behalf of each honest party $\mathcal{P}_j \in S_0$, Sim does the following:
  1. Generate and store a one-time key $K_j \leftarrow \{0,1\}^\lambda$.
  2. Compute a dummy ciphertext $\mathsf{ct}_{K_j} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, 0^\lambda)$.
  3. Compute a simulated proof $\pi_{K_j} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{K_j})$ for $\mathsf{st}_{K_j} = (\mathsf{ct}_{K_j}, \mathsf{pk}^{\mathsf{TFHE}}) \in L_1$.
  4. Generate a pair of signatures $(\sigma_{j,0}, \sigma_{j,1})$ as: $\sigma_{j,0} = \mathsf{Sign}(\mathsf{sk}_j, \mathsf{ct}_{\vec{\mathbf{u}},j})$, $\quad \sigma_{j,1} = \mathsf{Sign}(\mathsf{sk}_j, \mathsf{ct}_{K_j})$.
  5. Send the following to the party $\mathcal{P}^*$: $(\mathsf{ct}_{K_j}, \pi_{K_j}, \sigma_{j,0}, \sigma_{j,1})$.

- **Round 3: ($\rightarrow \mathsf{Sim}$)** On behalf of each honest party $\mathcal{P}_j \in S_0$, Sim receives the following from the corrupt party $\mathcal{P}^*$:

$$\{(\mathsf{ct}_{K_\ell,j}, \pi_{K_\ell,j}, \sigma_{\ell,j,0}, \sigma_{\ell,j,1})\}_{\mathcal{P}_\ell \in S}),$$

  and subsequently executes the following steps:
  1. **TFHE Ciphertext and Signature Validation:** For each honest party $\mathcal{P}_j \in S_0$, output $\perp$ and abort if any of the following events occur:
     1. There exists some party $\mathcal{P}_\ell \in S$ such that $\mathsf{Verify}(\pi_{K_\ell,j}, \mathsf{st}_{K_\ell,j}) \neq 1$ for language L_1 where $\mathsf{st}_{K_\ell,j} = (\mathsf{ct}_{K_\ell,j}, \mathsf{pk}^{\mathsf{TFHE}})$.
     2. There exists some party $\mathcal{P}_\ell \in S$ such that $\mathsf{Verify}(\mathsf{vk}_\ell, \mathsf{ct}_{\vec{\mathbf{u}},j}, \sigma_{\ell,j,0}) \neq 1$ (OR) $\mathsf{Verify}(\mathsf{vk}_\ell,\ \mathsf{ct}_{K_\ell,j}, \sigma_{\ell,j,1}) \neq 1$.
  2. **Routing-Consistency Check-2:** Check if there exists a pair of honest parties $(\mathcal{P}_j, \mathcal{P}_{j'})$ such that
  $$\{\mathsf{ct}_{K_\ell,j}\}_{\mathcal{P}_\ell \in S_0} \neq \{\mathsf{ct}_{K_\ell,j'}\}_{\mathcal{P}_\ell \in S_0}.$$
  If such a pair of honest parties exists, then abort by outputting "$\mathsf{SpecialAbort}$".
  3. Finally, if flag currently is set to 1, then abort by outputting "$\mathsf{SpecialAbort}$".

- **Round 4: ($\mathsf{Sim} \rightarrow$)** On behalf of each honest party $\mathcal{P}_j \in S_0$, Sim sends the following to the corrupt party $\mathcal{P}^*$:
$$(\mathsf{ct}_j, \{(\pi_{j,\ell}, \mu_{j,\ell})\}_{\mathcal{P}_\ell \in S}),$$
  by executing the following steps:
  1. For each corrupt party $\mathcal{P}_i \in S_1$, extract the corresponding one-time key K_i as:
  $$(K_i, \mathsf{r}_{K_i}) = \mathsf{NIZK.Ext}(\pi_{K_i}, \mathsf{st}_{K_i}),$$
  where $\mathsf{st}_{K_i} = (\mathsf{ct}_{K_i}, \mathsf{pk}^{\mathsf{TFHE}})$.
  2. Issue the following query to the ideal functionality $\mathcal{F}^{\mathsf{TS}}_{\text{FTT}}$: ("`SignOn`", $\mathsf{sid}, \mathsf{vk}, \mathsf{msg}, \mathcal{P}^*, \vec{\mathbf{u}}, S$) using he previously extracted measurement $\vec{\mathbf{u}}$.

3. Upon receipt of the messages $\{(\text{“}\mathtt{Signing\ Req}\text{”}, \mathsf{sid}, \mathsf{msg}, \mathcal{P}_\ell)\}_{\mathcal{P}_\ell \in S}$ from $\mathcal{F}^{\mathsf{TS}}_{\text{FTT}}$, respond with $(\text{“}\mathtt{Agreed}\text{”}, \mathsf{sid}, \mathsf{msg}, \mathcal{P}_j)$ on behalf of each honest party $\mathcal{P}_j \in S_0$.
4. If the ideal functionality $\mathcal{F}^{\mathsf{TS}}_{\text{FTT}}$ responds with $(\text{“}\mathtt{Parts}\text{”}, \mathsf{sid}, \overrightarrow{\mathbf{u}}, \mathsf{msg}, S, \{\mathsf{Token}_\ell\}_{\mathcal{P}_\ell \in S})$, then do the following:
    1. Compute the TFHE ciphertext $\mathsf{ct}_{\mathcal{C},\ell}$ corresponding to each $\mathcal{P}_\ell \in S$ as:
    $$\mathsf{ct}_{\mathcal{C},\ell} = \mathsf{TFHE.Eval}(\mathsf{pk}^{\mathsf{TFHE}}, \mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{K_\ell}).$$
    2. On behalf of each honest party $\mathcal{P}_j \in S_0$, simulate a partial decryption $\mu_{j,\ell}$ corresponding to each $\mathcal{P}_\ell \in S$ by invoking the following: $\mu_{j,\ell} = \mathsf{TFHE.Sim}(\mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{\mathcal{C}_\ell}, K_\ell)$.
    3. On behalf of each honest party $\mathcal{P}_j \in S_0$, compute the one-time encryption of the token as: $\mathsf{ct}_j = K_j \oplus \mathsf{Token}_j$.
5. Else, if the ideal functionality $\mathcal{F}^{\mathsf{TS}}_{\text{FTT}}$ responds with $(\text{“}\mathtt{SignOn\ failed}\text{”}, \mathsf{sid}, \overrightarrow{\mathbf{u}}, \mathsf{msg}, S)$, then do the following:
    1. Compute the TFHE ciphertext $\mathsf{ct}_{\mathcal{C},\ell}$ corresponding to each $\mathcal{P}_\ell \in S$ as:
    $$\mathsf{ct}_{\mathcal{C},\ell} = \mathsf{TFHE.Eval}(\mathsf{pk}^{\mathsf{TFHE}}, \mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{K_\ell}).$$
    2. On behalf of each honest party $\mathcal{P}_j \in S_0$, simulate a partial decryption $\mu_{j,\ell}$ corresponding to each $\mathcal{P}_\ell \in S$ by invoking the following: $\mu_{j,\ell} = \mathsf{TFHE.Sim}(\mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{\mathcal{C}_\ell}, 0^\lambda)$.
    3. On behalf of each honest party $\mathcal{P}_j \in S_0$, simulate the one-time encryption of the token as: $\mathsf{ct}_j \leftarrow \{0,1\}^\lambda$.
6. On behalf of each honest party $\mathcal{P}_j \in S_0$, compute a simulated proof $\pi_{j,\ell}$ corresponding to each $\mathcal{P}_\ell \in S$ as: $\pi_{j,\ell} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{j,\ell})$, where $\mathsf{st}_{j,\ell} = (\mathsf{ct}_{\mathcal{C},\ell}, \mu_{j,\ell}, \mathsf{com}_j)$.
7. Finally, on behalf of each honest party $\mathcal{P}_j \in S_0$, send $(\mathsf{ct}_j, \{(\pi_{j,\ell}, \mu_{j,\ell})\}_{\mathcal{P}_\ell \in S})$ to the corrupt party $\mathcal{P}^*$.

B.2.2 Hybrids

We now show that the above simulation strategy is successful against all environments $\mathcal{Z}$. That is, the view of the corrupt parties along with the output of the honest parties is computationally indistinguishable in the real and ideal worlds. We will show this 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 parties as in the real world.

2. Hyb_1 **- Special Abort:** In this hybrid, SimHyb acts exactly as in Hyb_0 except the following:

    1. **SignOn Phase: Case 2.** Suppose a corrupt party $\mathcal{P}^*$ initiates the sign-on phase. In this case, SimHyb outputs “$\mathsf{SpecialAbort}$” (as done by Sim in round 3 of the simulation strategy) if either of the following two scenarios occur:

   1. The adversary, acting on behalf of the initiating party, does not route the *same* TFHE ciphertext (encrypting the measurement $\overrightarrow{\mathbf{u}}$) to all the honest parties in the first round of the sign-on protocol, but all the corresponding signatures verify.
   2. The adversary, acting on behalf of the initiating party, does not route the *same* TFHE ciphertext (encrypting the one-time key K_j corresponding to some honest party $\mathcal{P}_j$) to all the honest parties in the third round of the sign-on protocol, but all the corresponding signatures verify.

3. Hyb_2 **- NIZK Extraction-1:** In this hybrid, SimHyb acts exactly as in Hyb_1 except the following:

   1. **SignOn Phase: Case 1.** Suppose an honest party $\mathcal{P}^*$ initiates the sign-on phase. SimHyb does the following as in the ideal world:
      1. **Round 3:** SimHyb does the following:
         1. For each corrupt party $\mathcal{P}_i \in S_1$, SimHyb extracts the one-time key K_i as:

            $$(K_i, \mathsf{r}_{K_i}) = \mathsf{NIZK.Ext}(\pi_{K_i}, \mathsf{st}_{K_i}),$$

            where $\mathsf{st}_{K_i} = (\mathsf{ct}_{K_i}, \mathsf{pk}^{\mathsf{TFHE}})$.

         2. If there exists some corrupt party $\mathcal{P}_i \in S_1$ such that $\mathsf{ct}_{K_i} \neq \mathsf{TFHE.Enc}$ $(\mathsf{pk}^{\mathsf{TFHE}}, K_i; \mathsf{r}_{K_i})$, SimHyb outputs $\perp$ and aborts.
      2. **Post-Round 4:** SimHyb does the following :
         1. For each corrupt party $\mathcal{P}_i \in S_1$, SimHyb extracts the TFHE partial decryption key $\mathsf{sk}_i^{\mathsf{TFHE}}$ as:

            $$(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{r}_i^{\mathsf{com}}) = \mathsf{NIZK.Ext}(\pi_{i,\ell}, \mathsf{st}_{i,\ell}),$$

            where $\mathsf{st}_{i,\ell} = (\mathsf{ct}_{\mathcal{C},\ell}, \mu_{i,\ell}, \mathsf{com}_i)$ and $\mathcal{P}_\ell$ is some party in the set S.

         2. If there exists some corrupt party $\mathcal{P}_i \in S_1$ such that $\mu_{i,\ell} \neq \mathsf{TFHE.PartialDec}$ $(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{ct}_{\mathcal{C},\ell})$ for some party $\mathcal{P}_\ell$, SimHyb outputs $\perp$ and aborts.

         3. Next, for each corrupt party $\mathcal{P}_i \in S_1$, SimHyb uses the previously extracted one-time key K_i to recover the corresponding partial token on the message msg as:

            $$\mathsf{Token}_i = K_i \oplus \mathsf{ct}_i.$$

         4. If there exists some corrupt party $\mathcal{P}_i \in S_1$ such that $\mathsf{Token}_i \neq \mathsf{TS.Sign}(\mathsf{sk}_i^{\mathsf{TS}}, \mathsf{msg})$, SimHyb outputs $\perp$ and aborts.

      3. **Switching Honest Party Outputs:** Finally, SimHyb no longer computes the output on behalf of honest parties as in the real world. Instead, as in the ideal world, SimHyb sends (“`Agreed`”, $\mathsf{sid}, \mathsf{msg}, \mathcal{P}_i$) on behalf of each corrupt party to the ideal functionality after validating the ciphertexts and tokens as done in the ideal world, and the honest parties receive output from the ideal functionality.

4. Hyb_3 **- NIZK Extraction-2:** In this hybrid, SimHyb acts exactly as in Hyb_2 except the following:

   1. **SignOn Phase: Case 2.** Suppose a corrupt party $\mathcal{P}^*$ initiates the sign-on phase. SimHyb does the following as in the ideal world:
      1. **<u>Round 1:</u>** SimHyb does the following:
         1. On behalf of each honest party $\mathcal{P}_j \in S_0$, receive a ciphertext $\mathsf{ct}_{\overrightarrow{\mathbf{u}},j}$ and a proof $\pi_{\overrightarrow{\mathbf{u}},j}$ from the party $\mathcal{P}^*$.
         2. Extract the corresponding measurement $\overrightarrow{\mathbf{u}}$ as:
         $$(\overrightarrow{\mathbf{u}}, \mathsf{r}_{\overrightarrow{\mathbf{u}},j}) = \mathsf{NIZK.Ext}(\pi_{\overrightarrow{\mathbf{u}},j}, \mathsf{st}_{\overrightarrow{\mathbf{u}},j}),$$
         where $\mathsf{st}_{\overrightarrow{\mathbf{u}},j} = (\mathsf{ct}_{\overrightarrow{\mathbf{u}},j}, \mathsf{pk}^{\mathsf{TFHE}})$.
         3. Output $\perp$ and abort if $\mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \overrightarrow{\mathbf{u}}; \mathsf{r}_{\overrightarrow{\mathbf{u}},j}) \neq \mathsf{ct}_{\overrightarrow{\mathbf{u}},j}$.
      2. **<u>Round 4:</u>** SimHyb does the following for each corrupt party $\mathcal{P}_i \in S_1$, :
         1. Extract the one-time key K_i as:
         $$(K_i, \mathsf{r}_{K_i}) = \mathsf{NIZK.Ext}(\pi_{K_i}, \mathsf{st}_{K_i}),$$
         where $\mathsf{st}_{K_i} = (\mathsf{ct}_{K_i}, \mathsf{pk}^{\mathsf{TFHE}})$.
         2. Output $\perp$ and abort if $\mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, K_i; \mathsf{r}_{K_i}) \neq \mathsf{ct}_{K_i}$.

5. Hyb_4 **- Simulate NIZK Proofs:** In this hybrid, SimHyb acts exactly as in Hyb_3 except the following:

   1. **SignOn Phase: Case 1.** Suppose an honest party $\mathcal{P}^*$ initiates the sign-on phase. SimHyb does the following as in the ideal world:
      1. **<u>Round 1:</u>** On behalf of party $\mathcal{P}^*$, SimHyb computes a simulated proof:
      $$\pi_{\overrightarrow{\mathbf{u}}} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{\overrightarrow{\mathbf{u}}}),$$
      where $\mathsf{st}_{\overrightarrow{\mathbf{u}}} = (\mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{pk}^{\mathsf{TFHE}}) \in L_1$.
      2. **<u>Round 3:</u>** On behalf of each honest party $\mathcal{P}_j$ in the set S_0, SimHyb computes a simulated proof:
      $$\pi_{K_j} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{K_j}),$$
      where $\mathsf{st}_{K_j} = (\mathsf{ct}_{K_j}, \mathsf{pk}^{\mathsf{TFHE}}) \in L_1$.
   2. **SignOn Phase: Case 2.** Suppose a corrupt party $\mathcal{P}^*$ initiates the sign-on phase. SimHyb does the following as in the ideal world:
      1. **<u>Round 2:</u>** On behalf of each honest party $\mathcal{P}_j$ in the set S_0, SimHyb computes a simulated proof:
      $$\pi_{K_j} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{K_j}),$$
      where $\mathsf{st}_{K_j} = (\mathsf{ct}_{K_j}, \mathsf{pk}^{\mathsf{TFHE}}) \in L_1$.
      2. **<u>Round 4:</u>** On behalf of each honest party $\mathcal{P}_j \in S_0$, SimHyb computes a simulated proof $\pi_{j,\ell}$ corresponding to each $\mathcal{P}_\ell \in S$ as:
      $$\pi_{j,\ell} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{j,\ell}),$$
      where $\mathsf{st}_{j,\ell} = (\mathsf{ct}_{\mathcal{C},\ell}, \mu_{j,\ell}, \mathsf{com}_j)$.

6. Hyb_5 **- Switch Commitments:** In this hybrid, SimHyb acts exactly as in Hyb_4 except the following: in the setup phase, corresponding to each honest party $\mathcal{P}_j \in S_0$, SimHyb computes the commitment to its TFHE threshold decryption key as:

$$\mathsf{com}_j = \mathsf{Commit}(0^\lambda; \mathsf{r}_j^{\mathsf{com}}),$$

as done in the ideal world.

7. Hyb_6 **- Simulate** TFHE **Partial Decryptions:** In this hybrid, SimHyb acts exactly as in Hyb_5 except the following:

   1. **SignOn Phase: Case 2.** Suppose a corrupt party $\mathcal{P}^*$ initiates the sign-on phase. SimHyb does the following in round 4 of the sign-on phase as in the ideal world:
      1. SimHyb issues the following query to the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ using the previously extracted measurement $\overrightarrow{\mathbf{u}}$:

         $$(\text{“}\mathtt{SignOn}\text{”}, \mathsf{sid}, \mathsf{vk}, \mathsf{msg}, \mathcal{P}^*, \overrightarrow{\mathbf{u}}, S).$$

         Upon receipt of the messages $\{(\text{“}\mathtt{Signing\ Req}\text{”}, \mathsf{sid}, \mathsf{msg}, \mathcal{P}_\ell)\}_{\mathcal{P}_\ell \in S}$ from $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$, it responds with $(\text{“}\mathtt{Agreed}\text{”}, \mathsf{sid}, \mathsf{msg}, \mathcal{P}_j)$ on behalf of each honest party $\mathcal{P}_j \in S_0$.
      2. Suppose that the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ responds with:

         $$(\text{“}\mathtt{Parts}\text{”}, \mathsf{sid}, \overrightarrow{\mathbf{u}}, \mathsf{msg}, S, \{\mathsf{Token}_\ell\}_{\mathcal{P}_\ell \in S}).$$

         On behalf of each honest party $\mathcal{P}_j \in S_0$, SimHyb simulates a partial decryption $\mu_{j,\ell}$ corresponding to each $\mathcal{P}_\ell \in S$ by invoking the following:

         $$\mu_{j,\ell} = \mathsf{TFHE.Sim}(\mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{K_\ell}, K_\ell).$$
      3. On the other hand, suppose that the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ responds with:

         $$(\text{“}\mathtt{SignOn\ failed}\text{”}, \mathsf{sid}, \overrightarrow{\mathbf{u}}, \mathsf{msg}, S).$$

         On behalf of each honest party $\mathcal{P}_j \in S_0$, SimHyb simulates a partial decryption $\mu_{j,\ell}$ corresponding to each $\mathcal{P}_\ell \in S$ by invoking the following:

         $$\mu_{j,\ell} = \mathsf{TFHE.Sim}(\mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{K_\ell}, 0^\lambda).$$

8. Hyb_7 **- Switch** TFHE **Ciphertexts:** In this hybrid, SimHyb acts exactly as in Hyb_6 except the following:

   1. **Registration Phase** SimHyb creates the following dummy encryption of the template as:

      $$\mathsf{ct}_{\overrightarrow{\mathbf{w}}} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \overrightarrow{\mathbf{w}}^*),$$

      where $\overrightarrow{\mathbf{w}}^*$ is any arbitrarily chosen template independent of the actual template $\overrightarrow{\mathbf{w}}$.
   2. **SignOn Phase: Case 1.** Suppose an honest party $\mathcal{P}^*$ initiates the sign-on phase. SimHyb does the following as in the ideal world:
      1. **<u>Round 1:</u>** On behalf of the initiating party $\mathcal{P}^*$, SimHyb computes a dummy encryption of the measurement as:

         $$\mathsf{ct}_{\overrightarrow{\mathbf{u}}} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \overrightarrow{\mathbf{u}}^*),$$

         where $\overrightarrow{\mathbf{u}}^*$ is any arbitrarily chosen measurement.

2. **<u>Round 3:</u>** On behalf of each honest party $\mathcal{P}_j \in S_0$, SimHyb computes a dummy TFHE encryption of the one-time encryption key K_j as:

$$\mathsf{ct}_{K_j} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, 0^\lambda).$$

3. **SignOn Phase: Case 2.** Suppose a corrupt party $\mathcal{P}^*$ initiates the sign-on phase. SimHyb does the following as in the ideal world: in round 3 of the sign-on phase, on behalf of each honest party $\mathcal{P}_j \in S_0$, SimHyb computes a dummy TFHE encryption of the one-time encryption key K_j as:

$$\mathsf{ct}_{K_j} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, 0^\lambda).$$

9. Hyb_8 **- Switch One-time Encryption Ciphertexts:** In this hybrid, SimHyb acts exactly as in Hyb_7 except the following:

   1. **SignOn Phase: Case 2.** Suppose a corrupt party $\mathcal{P}^*$ initiates the sign-on phase. SimHyb does the following in round 4 of the sign-on phase as in the ideal world:
      1. As in the previous hybrids, SimHyb uses the extracted measurement $\overrightarrow{\mathbf{u}}$ to query the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$.
      2. Suppose that the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ responds with:

      $$(\text{“}\mathtt{SignOn\ failed}\text{”}, \mathsf{sid}, \overrightarrow{\mathbf{u}}, \mathsf{msg}, S).$$

      On behalf of each honest party $\mathcal{P}_j \in S_0$, simulate the one-time encryption of the token as:

      $$\mathsf{ct}_j \leftarrow \{0,1\}^\lambda.$$

   Observe that this hybrid corresponds to the ideal world execution.

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

Lemma 6. *Assuming that the signature scheme is strongly existentially unforgeable,* Hyb_0 *is computationally indistinguishable from* Hyb_1*.*

Proof. The only difference between the two hybrids is that in Hyb_1, SimHyb might output "$\mathsf{SpecialAbort}$". We now show that SimHyb outputs "$\mathsf{SpecialAbort}$" in Hyb_1 only with negligible probability. Note that this immediately implies Lemma B.2.2.

Suppose not. That is, suppose there exists an environment $\mathcal{Z}$ that can cause SimHyb to output "$\mathsf{SpecialAbort}$" in Hyb_1 with non-negligible probability, then we will use $\mathcal{Z}$ to construct an adversary $\mathcal{A}_{\mathsf{Sign}}$ that breaks the strong unforgeability of the signature scheme with non-negligible probability, which is a contradiction.

$\mathcal{A}_{\mathsf{Sign}}$ begins an execution of the protocol interacting with the environment $\mathcal{Z}$ as in Hyb_0. Now, suppose $\mathcal{Z}$ corrupts a set of parties $\mathcal{P}_1$, and let $\mathcal{P}_0 = S \setminus S_1$ be the set of honest parties. For each honest party $\mathcal{P}_j \in S_0$, $\mathcal{A}_{\mathsf{Sign}}$ interacts with a challenger $\mathcal{C}_{\mathsf{Sign}}$ and gets a verification key vk_j which is forwarded to $\mathcal{Z}$ as part of the setup phase of DiFuz. Then, during the course of the protocol, $\mathcal{A}_{\mathsf{Sign}}$ forwards signature queries corresponding to honest parties from $\mathcal{Z}$ to $\mathcal{C}_{\mathsf{Sign}}$ and the responses from $\mathcal{C}_{\mathsf{Sign}}$ to $\mathcal{Z}$.

Finally, suppose $\mathcal{Z}$ causes $\mathcal{A}_{\mathsf{Sign}}$ to output "$\mathsf{SpecialAbort}$" with non-negligible probability. This in turn implies that one of the following cases must occur with non-negligible probability:

1. There exists a pair of honest parties $(\mathcal{P}_j, \mathcal{P}_{j'})$ in the set of honest parties S_0 such that

$$\mathsf{ct}_{\overrightarrow{\mathbf{u}},j} \neq \mathsf{ct}_{\overrightarrow{\mathbf{u}},j'}.$$

   Note that $\mathcal{Z}$ is allowed to forward exactly $|S_0|$-many signature queries on the same measurement ciphertext $\mathsf{ct}_{\overrightarrow{\mathbf{u}}}$ (one query corresponding to each honest party). This in turn implies that at least one of the two signatures $\sigma_{j,j,0}$ (on message $\mathsf{ct}_{\overrightarrow{\mathbf{u}},j}$) and $\sigma_{j',j',0}$ (on message $\mathsf{ct}_{\overrightarrow{\mathbf{u}},j'}$) produced by $\mathcal{Z}$ was not received from the challenger $\mathcal{C}_{\mathsf{Sign}}$.

   We assume without loss of generality that the signature $\sigma_{j',j',0}$ was not produced as a result of a query to the challenger $\mathcal{C}_{\mathsf{Sign}}$. Note that since $\mathcal{A}_{\mathsf{Sign}}$ outputs "$\mathsf{SpecialAbort}$", it follows that this signature verifies correctly, i.e., we have:

$$\mathsf{Verify}(\mathsf{vk}_{j'}, \mathsf{ct}_{\overrightarrow{\mathbf{u}},j'}, \sigma_{j',j',0}) = 1.$$

   Then $\mathcal{A}_{\mathsf{Sign}}$ can output the tuple $(\mathsf{ct}_{\overrightarrow{\mathbf{u}},j'}, \sigma_{j',j',0})$ as a valid forgery to break the strong unforgeability of the signature scheme.

2. There exists a pair of honest parties $(\mathcal{P}_j, \mathcal{P}_{j'})$ in the set of honest parties S_0 such that

$$\{\mathsf{ct}_{K_\ell,j}\}_{\mathcal{P}_\ell \in S_0} \neq \{\mathsf{ct}_{K_\ell,j'}\}_{\mathcal{P}_\ell \in S_0}.$$

   Note that for a given honest party $\mathcal{P}_\ell \in S_0$, $\mathcal{Z}$ is allowed to forward exactly $|S_0|$-many signature queries on ct_{K_ℓ} (one query corresponding to each honest party). This in turn implies that, without loss of generality, there exists a signature $\sigma_{\ell,j',1}$ (on message $\mathsf{ct}_{K_\ell,j'}$) produced by $\mathcal{Z}$ that was not received from the challenger $\mathcal{C}_{\mathsf{Sign}}$.

   In addition, since $\mathcal{A}_{\mathsf{Sign}}$ outputs "$\mathsf{SpecialAbort}$", it follows that this signature verifies correctly, i.e., we have:

$$\mathsf{Verify}(\mathsf{vk}_\ell, \mathsf{ct}_{K_\ell,j'}, \sigma_{\ell,j',1}) = 1.$$

   Then $\mathcal{A}_{\mathsf{Sign}}$ can output the tuple $(\mathsf{ct}_{K_\ell,j'}, \sigma_{\ell,j',1})$ as a valid forgery to break the strong unforgeability of the signature scheme.

In summary, if there exists an environment $\mathcal{Z}$ that can cause SimHyb to output "$\mathsf{SpecialAbort}$" in Hyb_1 with non-negligible probability, then there exists an algorithm $\mathcal{A}_{\mathsf{Sign}}$ that breaks the strong unforgeability of the signature scheme with non-negligible probability, which is a contradiction.

□

Lemma 7. *Assuming that the NIZK argument system is simulation extractable, the commitment scheme is computationally binding, and the TFHE scheme satisfies decryption correctness,* Hyb_1 *is computationally indistinguishable from* Hyb_2*.*

Proof. Suppose there exists an environment $\mathcal{Z}$ that can distinguish between Hyb_1 and Hyb_2 with non-negligible probability. Thus implies that with non-negligible probability, in Hyb_2, SimHyb sends the "agreed" message to the ideal functionality even if the honest parties are unable to compute the token by running the actual output computation phase in Hyb_1. This in turn implies that one of the following cases must be true when an honest party $\mathcal{P}^*$ initiates the sign-on protocol:

1. **Case-1:** In round 3 of the sign-on phase, some corrupt party $\mathcal{P}_i \in S_1$ generates an invalid ciphertext ct_{K_i} and a corresponding NIZK proof, but with non-negligible probability, SimHyb fails to detect this via NIZK extraction and hence does not abort. This immediately implies the existence of an algorithm that can break the simulation extractability of the NIZK argument system with non-negligible probability, which is a contradiction.

2. **Case-2:** In round 4 of the sign-on phase, some corrupt party $\mathcal{P}_i \in S_1$ generates an invalid partial decryption $\mu_{i,\ell}$ and a corresponding NIZK proof, but with non-negligible probability, SimHyb fails to detect this via NIZK extraction and hence does not abort. This immediately implies the existence of an algorithm that can break the simulation extractability of the NIZK argument system with non-negligible probability, which is a contradiction.

3. **Case-3:** In round 4 of the sign-on phase, some corrupt party $\mathcal{P}_i \in S_1$ generates a partial decryption $\mu_{i,\ell}$ and a corresponding NIZK proof that is invalid with respect to the witness $(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{r}_i)$, but $\mathcal{P}_i$ produces a dummy witness $(\widetilde{\mathsf{sk}_i^{\mathsf{TFHE}}}, \widetilde{\mathsf{r}_i^{\mathsf{com}}}) \neq (\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{r}_i^{\mathsf{com}})$ that validates the NIZK proof *and* produces the same commitment com_i, i.e., we have

$$\mathsf{Commit}(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{r}_i^{\mathsf{com}}) = \mathsf{Commit}(\widetilde{\mathsf{sk}_i^{\mathsf{TFHE}}}, \widetilde{\mathsf{r}_i^{\mathsf{com}}}) = \mathsf{com}_i.$$

   This immediately implies the existence of an algorithm that can break the computationally binding property of the commitment scheme with non-negligible probability, which is a contradiction.

4. **Case-4:** In round 4 of the sign-on phase, some corrupt party $\mathcal{P}_i \in S_1$ generates an invalid token Token_i, but SimHyb fails to detect this. This is only possible if the NIZK extraction procedure produces, with non-negligible probability, a witness $(\widetilde{K_i}, \widetilde{\mathsf{r}_{K_i}}) \neq (K_i, \mathsf{r}_{K_i})$. Assuming that the TFHE scheme produces correct decryptions, this immediately implies the existence of an algorithm that can break the simulation extractability of the NIZK argument system with non-negligible probability, which is a contradiction.

Thus, except with negligible probability, in Hyb_2, SimHyb does not send the "agreed" message to the ideal functionality whenever the honest party is unable to compute the token by running the actual output computation phase in Hyb_1. This completes the proof of the lemma.

$\square$

Lemma 8. *Assuming that the NIZK argument system is simulation extractable, the commitment scheme is computationally binding, and the TFHE scheme satisfies decryption correctness,* Hyb_2 *is computationally indistinguishable from* Hyb_3.

Proof. Suppose there exists an environment $\mathcal{Z}$ that can distinguish between Hyb_2 and Hyb_3 with non-negligible probability. This implies that one of the following cases must be true when a corrupt party $\mathcal{P}^*$ initiates the sign-on protocol:

1. **Case-1:** In round 1 of the sign-on phase, the corrupt initiator $\mathcal{P}^*$ generates an invalid measurement ciphertext $\mathsf{ct}_{\overrightarrow{\mathbf{u}}}$ and a corresponding NIZK proof, but with non-negligible probability, SimHyb fails to detect this via NIZK extraction and hence does not abort. This immediately implies the existence of an algorithm that can break the simulation

extractability of the NIZK argument system with non-negligible probability, which is a contradiction.

2. **Case-2:** In round 4 of the sign-on phase, some corrupt party $\mathcal{P}_i \in S_1$ generates an invalid partial decryption $\mu_{i,\ell}$ and a corresponding NIZK proof, but with non-negligible probability, SimHyb fails to detect this via NIZK extraction and hence does not abort. This immediately implies the existence of an algorithm that can break the simulation extractability of the NIZK argument system with non-negligible probability, which is a contradiction.

3. **Case-3:** In round 4 of the sign-on phase, some corrupt party $\mathcal{P}_i \in S_1$ generates a partial decryption $\mu_{i,\ell}$ and a corresponding NIZK proof that is invalid with respect to the witness $(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{r}_i)$, but $\mathcal{P}_i$ produces a dummy witness $(\widetilde{\mathsf{sk}_i^{\mathsf{TFHE}}}, \widetilde{\mathsf{r}_i^{\mathsf{com}}}) \neq (\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{r}_i^{\mathsf{com}})$ that validates the NIZK proof *and* produces the same commitment com_i, i.e., we have

$$\mathsf{Commit}(\mathsf{sk}_i^{\mathsf{TFHE}}, \mathsf{r}_i^{\mathsf{com}}) = \mathsf{Commit}(\widetilde{\mathsf{sk}_i^{\mathsf{TFHE}}}, \widetilde{\mathsf{r}_i^{\mathsf{com}}}) = \mathsf{com}_i.$$

This immediately implies the existence of an algorithm that can break the computationally binding property of the commitment scheme with non-negligible probability, which is a contradiction.

This completes the proof of the lemma.

□

Lemma 9. *Assuming that the NIZK argument system satisfies the zero knowledge property,* Hyb_3 *is computationally indistinguishable from* Hyb_4*.*

Proof. The only difference between the two hybrids is that in Hyb_3, SimHyb computes the messages of the NIZK argument system by running the honest prover algorithm $\mathsf{Prove}(\cdot)$, while in Hyb_4, they are computed by running the simulator $\mathsf{NIZK.Sim}(\cdot)$.

Suppose there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability. We will use $\mathcal{Z}$ to construct an adversary $\mathcal{A}_{\mathsf{NIZK}}$ that breaks the zero-konwledge property of the NIZK argumnt system with non-negligible probability, which is a contradiction.

$\mathcal{A}_{\mathsf{NIZK}}$ begins an execution of the fuzzy threshold token generation protocol $\pi^{\mathsf{Any-TFHE}}$ interacting with the environment $\mathcal{Z}$. At the same time, $\mathcal{A}_{\mathsf{NIZK}}$ begins interacting with a challenger $\mathcal{C}_{\mathsf{NIZK}}$ in a zero-knowledge experiment against the NIZK argument system. Now, suppose $\mathcal{Z}$ corrupts a set of parties $\mathcal{P}_1$, and let $\mathcal{P}_0 = \mathcal{P} \setminus \mathcal{P}_1$ be the set of honest parties.

The challenger $\mathcal{C}_{\mathsf{NIZK}}$ uniformly samples a bit $b \leftarrow \{0,1\}$. $\mathcal{A}_{\mathsf{NIZK}}$ executes the simulation exactly as done by SimHyb in hybrid Hyb_3, except the following:

1. **SignOn Phase: Case 1.** Suppose an honest party $\mathcal{P}^*$ initiates the sign-on phase. $\mathcal{A}_{\mathsf{NIZK}}$ does the following:
   1. **<u>Round 1:</u>** On behalf of party $\mathcal{P}^*$, $\mathcal{A}_{\mathsf{NIZK}}$ computes a proof $\pi_{\vec{\mathbf{u}}}$ as follows:
      1. $\mathcal{A}_{\mathsf{TFHE}}$ queries the challenger $\mathcal{C}_{\mathsf{NIZK}}$ with the tuple $(\mathsf{st}_{\vec{\mathbf{u}}}, \mathsf{wit}_{\vec{\mathbf{u}}})$, where

$$\mathsf{st}_{\vec{\mathbf{u}}} = (\mathsf{ct}_{\vec{\mathbf{u}}}, \mathsf{pk}^{\mathsf{TFHE}}) \in L_1, \quad \mathsf{wit}_{\vec{\mathbf{u}}} = (\vec{\mathbf{u}}, \mathsf{r}_{\vec{\mathbf{u}}}).$$

2. If $b = 0$, the challenger $\mathcal{C}_{\mathsf{NIZK}}$ responds with

$$\pi_{\overrightarrow{\mathbf{u}}} \leftarrow \mathsf{Prove}(\mathsf{st}_{\overrightarrow{\mathbf{u}}}, \mathsf{wit}_{\overrightarrow{\mathbf{u}}}).$$

3. If $b = 1$, the challenger $\mathcal{C}_{\mathsf{NIZK}}$ responds with

$$\pi_{\overrightarrow{\mathbf{u}}} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{\overrightarrow{\mathbf{u}}}).$$

4. Now, $\mathcal{A}_{\mathsf{NIZK}}$ uses the same proof $\pi_{\overrightarrow{\mathbf{u}}}$ received from $\mathcal{C}_{\mathsf{TFHE}}$ for the simulation.

2. **<u>Round 3:</u>** On behalf of each honest party $\mathcal{P}_j$ in the set S_0, $\mathcal{A}_{\mathsf{NIZK}}$ computes a proof π_{K_j} as follows:
   1. $\mathcal{A}_{\mathsf{TFHE}}$ queries the challenger $\mathcal{C}_{\mathsf{NIZK}}$ with the tuple $(\mathsf{st}_{K_j}, \mathsf{wit}_{K_j})$, where

   $$\mathsf{st}_{K_j} = (\mathsf{ct}_{K_j}, \mathsf{pk}^{\mathsf{TFHE}}) \in L_1, \quad \mathsf{wit}_{K_j} = (K_j, \mathsf{r}_{K_j}).$$

   2. If $b = 0$, the challenger $\mathcal{C}_{\mathsf{NIZK}}$ responds with

   $$\pi_{K_j} \leftarrow \mathsf{Prove}(\mathsf{st}_{K_j}, \mathsf{wit}_{K_j}).$$

   3. If $b = 1$, the challenger $\mathcal{C}_{\mathsf{NIZK}}$ responds with

   $$\pi_{K_j} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{K_j}).$$

   4. Now, $\mathcal{A}_{\mathsf{NIZK}}$ uses the same proof π_{K_j} received from $\mathcal{C}_{\mathsf{TFHE}}$ for the simulation.

2. **SignOn Phase: Case 2.** Suppose a corrupt party $\mathcal{P}^*$ initiates the sign-on phase. $\mathcal{A}_{\mathsf{NIZK}}$ does the following:

   1. **<u>Round 3:</u>** On behalf of each honest party $\mathcal{P}_j$ in the set S_0, $\mathcal{A}_{\mathsf{NIZK}}$ computes a proof π_{K_j} as follows:
      1. $\mathcal{A}_{\mathsf{TFHE}}$ queries the challenger $\mathcal{C}_{\mathsf{NIZK}}$ with the tuple $(\mathsf{st}_{K_j}, \mathsf{wit}_{K_j})$, where

      $$\mathsf{st}_{K_j} = (\mathsf{ct}_{K_j}, \mathsf{pk}^{\mathsf{TFHE}}) \in L_1, \quad \mathsf{wit}_{K_j} = (K_j, \mathsf{r}_{K_j}).$$

      2. If $b = 0$, the challenger $\mathcal{C}_{\mathsf{NIZK}}$ responds with

      $$\pi_{K_j} \leftarrow \mathsf{Prove}(\mathsf{st}_{K_j}, \mathsf{wit}_{K_j}).$$

      3. If $b = 1$, the challenger $\mathcal{C}_{\mathsf{NIZK}}$ responds with

      $$\pi_{K_j} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{K_j}).$$

      4. Now, $\mathcal{A}_{\mathsf{NIZK}}$ uses the same proof π_{K_j} received from $\mathcal{C}_{\mathsf{TFHE}}$ for the simulation.

   2. **<u>Round 4:</u>** On behalf of each honest party $\mathcal{P}_j$ in the set S_0, $\mathcal{A}_{\mathsf{NIZK}}$ computes a proof $\pi_{j,\ell}$ corresponding to each $\mathcal{P}_\ell \in S$ as follows:
      1. $\mathcal{A}_{\mathsf{TFHE}}$ queries the challenger $\mathcal{C}_{\mathsf{NIZK}}$ with the tuple $(\mathsf{st}_{j,\ell}, \mathsf{wit}_j)$, where

      $$\mathsf{st}_{j,\ell} = (\mathsf{ct}_{\mathcal{C},\ell}, \mu_{j,\ell}, \mathsf{com}_j) \in L_2, \quad \mathsf{wit}_j = (\mathsf{sk}_j^{\mathsf{TFHE}}, \mathsf{r}_j^{\mathsf{com}}).$$

      2. If $b = 0$, the challenger $\mathcal{C}_{\mathsf{NIZK}}$ responds with

      $$\pi_{j,\ell} \leftarrow \mathsf{Prove}(\mathsf{st}_{j,\ell}, \mathsf{wit}_j).$$

3. If $b = 1$, the challenger $\mathcal{C}_{\mathsf{NIZK}}$ responds with

$$\pi_{j,\ell} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{j,\ell}).$$

4. Now, $\mathcal{A}_{\mathsf{NIZK}}$ uses the same proof $\pi_{j,\ell}$ received from $\mathcal{C}_{\mathsf{TFHE}}$ for the simulation.

$\mathcal{A}_{\mathsf{NIZK}}$ now continues with the rest of the simulation as in Hyb_3. Notice that when the bit b chosen by the challenger $\mathcal{C}_{\mathsf{NIZK}}$ is 0, the simulation experiment between $\mathcal{A}_{\mathsf{NIZK}}$ and $\mathcal{Z}$ corresponds exactly to Hyb_3. On the other hand, when the bit b chosen by the challenger $\mathcal{C}_{\mathsf{NIZK}}$ is 1, the experiment between $\mathcal{A}_{\mathsf{NIZK}}$ and $\mathcal{Z}$ corresponds exactly to Hyb_4.

Thus, if the environment $\mathcal{Z}$ can distinguish between the two hybrids with non-negligible probability, then the algorithm $\mathcal{A}_{\mathsf{NIZK}}$ can break the zero-konwledge property of the NIZK argumnt system with non-negligible probability, which is a contradiction.

□

Remark: We note that since the NIZK argument system is simulation extractable, the values extracted by the simulator from the adversary's proofs (in Hyb_2) do not change even the simulator is computing simulated proofs in Hyb_3 instead of generating the proofs honestly.

Lemma 10. *Assuming that the commitment scheme is computationally hiding,* Hyb_4 *is computationally indistinguishable from* Hyb_5*.*

Proof. The only difference between the two hybrids is that in Hyb_4, corresponding to each honest party $\mathcal{P}_j \in S_0$, SimHyb computes the commitment to its TFHE threshold decryption key as:

$$\mathsf{com}_j = \mathsf{Commit}(\mathsf{sk}_j^{\mathsf{TFHE}}; \mathsf{r}_j^{\mathsf{com}}),$$

as done in the real world, while in Hyb_4, corresponding to each honest party $\mathcal{P}_j \in S_0$, SimHyb computes the commitment to its TFHE threshold decryption key as:

$$\mathsf{com}_j = \mathsf{Commit}(\mathsf{sk}_j^{\mathsf{TFHE}}; \mathsf{r}_j^{\mathsf{com}}),$$

as done in the ideal world.

Suppose there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability. We will use $\mathcal{Z}$ to construct an adversary $\mathcal{A}_{\mathsf{com}}$ that breaks the hiding property of the commitment scheme with non-negligible probability, which is a contradiction.

$\mathcal{A}_{\mathsf{com}}$ begins an execution of the fuzzy threshold token generation protocol $\pi^{\mathsf{Any-TFHE}}$ interacting with the environment $\mathcal{Z}$. At the same time, $\mathcal{A}_{\mathsf{com}}$ begins interacting with a challenger $\mathcal{C}_{\mathsf{com}}$ in a hiding security experiment against the commitment scheme. Now, suppose $\mathcal{Z}$ corrupts a set of parties $\mathcal{P}_1$, and let $\mathcal{P}_0 = \mathcal{P} \setminus \mathcal{P}_1$ be the set of honest parties.

The challenger $\mathcal{C}_{\mathsf{com}}$ uniformly samples a bit $b \leftarrow \{0,1\}$. For each honest party $\mathcal{P}_j \in S_0$, $\mathcal{A}_{\mathsf{com}}$ sends the corresponding TFHE decryption key $\mathsf{sk}_j^{\mathsf{TFHE}}$ to the challenger $\mathcal{C}_{\mathsf{com}}$. In response, the challenger responds with a commitment com_j where:

1. If $b = 0$, the challenger $\mathcal{C}_{\mathsf{com}}$ sets

$$\mathsf{com}_j = \mathsf{Commit}(\mathsf{sk}_j^{\mathsf{TFHE}}; \mathsf{r}_j^{\mathsf{com}}).$$

2. If $b = 1$, the challenger $\mathcal{C}_{\mathsf{com}}$ sets

$$\mathsf{com}_j = \mathsf{Commit}(0^\lambda; \mathsf{r}_j^{\mathsf{com}})$$

$\mathcal{A}_{\mathsf{com}}$ now continues with the rest of the simulation as in Hyb_4 using the commitments produced by the the challenger $\mathcal{C}_{\mathsf{com}}$. Note that since all NIZK proofs are simulated in Hyb_4, the simulation does not require $\mathcal{A}_{\mathsf{com}}$ to have knowledge of the randomness $\mathsf{r}_j^{\mathsf{com}}$ used by $\mathcal{C}_{\mathsf{com}}$ to generate each commitment com_j.

Now, observe that when the bit b chosen by the challenger $\mathcal{C}_{\mathsf{com}}$ is 0, the simulation experiment between $\mathcal{A}_{\mathsf{com}}$ and $\mathcal{Z}$ corresponds exactly to Hyb_4. On the other hand, when the bit b chosen by the challenger $\mathcal{C}_{\mathsf{com}}$ is 1, the experiment between $\mathcal{A}_{\mathsf{com}}$ and $\mathcal{Z}$ corresponds exactly to Hyb_5.

Thus, if the environment $\mathcal{Z}$ can distinguish between the two hybrids with non-negligible probability, then the algorithm $\mathcal{A}_{\mathsf{com}}$ can break the hiding security of the commitment scheme with non-negligible probability, which is a contradiction.

□

Lemma 11. *Assuming that the* TFHE *scheme satisfies simulation security,* Hyb_5 *is computationally indistinguishable from* Hyb_6.

Proof. Suppose there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability. We will use $\mathcal{Z}$ to construct an adversary $\mathcal{A}_{\mathsf{TFHE}}$ that breaks the simulation security of the TFHE scheme with non-negligible probability, which is a contradiction.

$\mathcal{A}_{\mathsf{TFHE}}$ begins an execution of the fuzzy threshold token generation protocol $\pi^{\mathsf{Any-TFHE}}$ interacting with the environment $\mathcal{Z}$. At the same time, $\mathcal{A}_{\mathsf{TFHE}}$ begins interacting with a challenger $\mathcal{C}_{\mathsf{TFHE}}$ in a simulation security experiment against the commitment scheme. Now, suppose $\mathcal{Z}$ corrupts a set of parties $\mathcal{P}_1$, and let $\mathcal{P}_0 = \mathcal{P} \setminus \mathcal{P}_1$ be the set of honest parties. In the TFHE experiment, $\mathcal{A}_{\mathsf{TFHE}}$ corrupts the same set of parties by querying their respective partial decryption keys.

The challenger $\mathcal{C}_{\mathsf{TFHE}}$ uniformly samples a bit $b \leftarrow \{0,1\}$. Now suppose a corrupt party $\mathcal{P}^*$ initiates the sign-on phase. $\mathcal{A}_{\mathsf{TFHE}}$ executes the simulation exactly as done by SimHyb in hybrid Hyb_5, except in round 4 of the sign-on phase, where it does the following:

1. $\mathcal{A}_{\mathsf{TFHE}}$ uses the extracted measurement $\overrightarrow{\mathbf{u}}$ to query the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$.
2. Suppose that the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ responds with:

$$(\text{“}\texttt{Parts}\text{”}, \mathsf{sid}, \overrightarrow{\mathbf{u}}, \mathsf{msg}, S, \{\mathsf{Token}_\ell\}_{\mathcal{P}_\ell \in S}).$$

   On behalf of each honest party $\mathcal{P}_j \in S_0$, $\mathcal{A}_{\mathsf{TFHE}}$ simulates a partial decryption $\mu_{j,\ell}$ corresponding to each $\mathcal{P}_\ell \in S$ as follows:

   1. $\mathcal{A}_{\mathsf{TFHE}}$ queries the challenger $\mathcal{C}_{\mathsf{TFHE}}$ with the tuple $(\mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{K_\ell}, K_\ell)$.
   2. If $b = 0$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mu_{j,\ell} = \mathsf{TFHE.PartialDec}(\mathsf{sk}_j^{\mathsf{TFHE}}, \mathsf{TFHE.Eval}(\mathsf{pk}^{\mathsf{TFHE}}, \mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{K_\ell})).$$

   3. if $b = 1$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mu_{j,\ell} = \mathsf{TFHE.Sim}(\mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{K_\ell}, K_\ell).$$

   4. Now, $\mathcal{A}_{\mathsf{TFHE}}$ uses the same partial decryption $\mu_{j,\ell}$ received from $\mathcal{C}_{\mathsf{TFHE}}$ for the simulation.

3. On the other hand, suppose that the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ responds with:

$$(\text{“}\texttt{SignOn failed}\text{”}, \mathsf{sid}, \overrightarrow{\mathbf{u}}, \mathsf{msg}, S).$$

On behalf of each honest party $\mathcal{P}_j \in S_0$, $\mathcal{A}_{\mathsf{TFHE}}$ simulates a partial decryption $\mu_{j,\ell}$ corresponding to each $\mathcal{P}_\ell \in S$ as follows:

   1. $\mathcal{A}_{\mathsf{TFHE}}$ queries the challenger $\mathcal{C}_{\mathsf{TFHE}}$ with the tuple $(\mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{K_\ell}, 0^\lambda)$.
   2. If $b = 0$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mu_{j,\ell} = \mathsf{TFHE.PartialDec}(\mathsf{sk}_j^{\mathsf{TFHE}}, \mathsf{TFHE.Eval}(\mathsf{pk}^{\mathsf{TFHE}}, \mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{K_\ell})).$$

   3. if $b = 1$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mu_{j,\ell} = \mathsf{TFHE.Sim}(\mathcal{C}_{\mathsf{Dist}}, \mathsf{ct}_{\overrightarrow{\mathbf{w}}}, \mathsf{ct}_{\overrightarrow{\mathbf{u}}}, \mathsf{ct}_{K_\ell}, 0^\lambda).$$

   4. Once again, $\mathcal{A}_{\mathsf{TFHE}}$ uses the same partial decryption $\mu_{j,\ell}$ received from $\mathcal{C}_{\mathsf{TFHE}}$ for the simulation.

$\mathcal{A}_{\mathsf{TFHE}}$ now continues with the rest of the simulation as in Hyb_5. Notice that when the bit b chosen by the challenger $\mathcal{C}_{\mathsf{TFHE}}$ is 0, the simulation experiment between $\mathcal{A}_{\mathsf{TFHE}}$ and $\mathcal{Z}$ corresponds exactly to Hyb_5. On the other hand, when the bit b chosen by the challenger $\mathcal{C}_{\mathsf{TFHE}}$ is 1, the experiment between $\mathcal{A}_{\mathsf{TFHE}}$ and $\mathcal{Z}$ corresponds exactly to Hyb_6.

Thus, if the environment $\mathcal{Z}$ can distinguish between the two hybrids with non-negligible probability, then the algorithm $\mathcal{A}_{\mathsf{TFHE}}$ can break the simulation security of the TFHE scheme with non-negligible probability, which is a contradiction.

□

Lemma 12. *Assuming that the* TFHE *scheme satisfies semantic security,* Hyb_6 *is computationally indistinguishable from* Hyb_7*.*

Proof. Suppose there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability. We will use $\mathcal{Z}$ to construct an adversary $\mathcal{A}_{\mathsf{TFHE}}$ that breaks the simulation security of the TFHE scheme with non-negligible probability, which is a contradiction.

$\mathcal{A}_{\mathsf{TFHE}}$ begins an execution of the fuzzy threshold token generation protocol $\pi^{\mathsf{Any-TFHE}}$ interacting with the environment $\mathcal{Z}$. At the same time, $\mathcal{A}_{\mathsf{TFHE}}$ begins interacting with a challenger $\mathcal{C}_{\mathsf{TFHE}}$ in a simulation security experiment against the commitment scheme. Now, suppose $\mathcal{Z}$ corrupts a set of parties $\mathcal{P}_1$, and let $\mathcal{P}_0 = \mathcal{P} \setminus \mathcal{P}_1$ be the set of honest parties. In the TFHE experiment, $\mathcal{A}_{\mathsf{TFHE}}$ corrupts the same set of parties by querying their respective partial decryption keys.

The challenger $\mathcal{C}_{\mathsf{TFHE}}$ uniformly samples a bit $b \leftarrow \{0,1\}$. $\mathcal{A}_{\mathsf{TFHE}}$ executes the simulation exactly as done by SimHyb in hybrid Hyb_6, except the following:

1. **Registration Phase.** $\mathcal{A}_{\mathsf{TFHE}}$ creates an encryption of the template as follows:

   1. $\mathcal{A}_{\mathsf{TFHE}}$ queries the challenger $\mathcal{C}_{\mathsf{TFHE}}$ with the tuple $(\overrightarrow{\mathbf{w}}, \overrightarrow{\mathbf{w}}^*)$, where $\overrightarrow{\mathbf{w}}^*$ is any arbitrarily chosen template independent of the actual template $\overrightarrow{\mathbf{w}}$.
   2. If $b = 0$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mathsf{ct}_{\overrightarrow{\mathbf{w}}} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \overrightarrow{\mathbf{w}}).$$

3. if $b = 1$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mathsf{ct}_{\vec{\mathbf{w}}} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \vec{\mathbf{w}}^*).$$

4. Now, $\mathcal{A}_{\mathsf{TFHE}}$ uses the same ciphertext $\mathsf{ct}_{\vec{\mathbf{w}}}$ received from $\mathcal{C}_{\mathsf{TFHE}}$ for the simulation.

2. **SignOn Phase: Case 1.** Suppose an honest party $\mathcal{P}^*$ initiates the sign-on phase. $\mathcal{A}_{\mathsf{TFHE}}$ does the following:

   1. **Round 1:** On behalf of the initiating party $\mathcal{P}^*$, SimHyb computes an encryption of the measurement as follows:
      1. $\mathcal{A}_{\mathsf{TFHE}}$ queries the challenger $\mathcal{C}_{\mathsf{TFHE}}$ with the tuple $(\vec{\mathbf{u}}, \vec{\mathbf{u}}^*)$, where $\vec{\mathbf{u}}^*$ is any arbitrarily chosen template independent of the actual template $\vec{\mathbf{u}}$.
      2. If $b = 0$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mathsf{ct}_{\vec{\mathbf{u}}} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \vec{\mathbf{u}}).$$

      3. if $b = 1$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mathsf{ct}_{\vec{\mathbf{u}}} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, \vec{\mathbf{u}}^*).$$

      4. Now, $\mathcal{A}_{\mathsf{TFHE}}$ uses the same ciphertext $\mathsf{ct}_{\vec{\mathbf{u}}}$ received from $\mathcal{C}_{\mathsf{TFHE}}$ for the simulation.
   2. **Round 3:** On behalf of each honest party $\mathcal{P}_j \in S_0$, $\mathcal{A}_{\mathsf{TFHE}}$ computes an encryption of the one-time encryption key K_j as follows:
      1. $\mathcal{A}_{\mathsf{TFHE}}$ queries the challenger $\mathcal{C}_{\mathsf{TFHE}}$ with the tuple $(K_j, 0^\lambda)$.
      2. If $b = 0$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mathsf{ct}_{K_j} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, K_j).$$

      3. if $b = 1$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mathsf{ct}_{K_j} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, 0^\lambda).$$

      4. Now, $\mathcal{A}_{\mathsf{TFHE}}$ uses the same ciphertext ct_{K_j} received from $\mathcal{C}_{\mathsf{TFHE}}$ for the simulation.

3. **SignOn Phase: Case 2.** Suppose a corrupt party $\mathcal{P}^*$ initiates the sign-on phase. In round 3 of the sign-on phase, on behalf of each honest party $\mathcal{P}_j \in S_0$, $\mathcal{A}_{\mathsf{TFHE}}$ computes an encryption of the one-time encryption key K_j as follows:

   1. $\mathcal{A}_{\mathsf{TFHE}}$ queries the challenger $\mathcal{C}_{\mathsf{TFHE}}$ with the tuple $(K_j, 0^\lambda)$.
   2. If $b = 0$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mathsf{ct}_{K_j} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, K_j).$$

   3. if $b = 1$, the challenger $\mathcal{C}_{\mathsf{TFHE}}$ responds with

$$\mathsf{ct}_{K_j} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, 0^\lambda).$$

   4. Now, $\mathcal{A}_{\mathsf{TFHE}}$ uses the same ciphertext ct_{K_j} received from $\mathcal{C}_{\mathsf{TFHE}}$ for the simulation.

$\mathcal{A}_{\mathsf{TFHE}}$ now continues with the rest of the simulation as in Hyb_6. Notice that when the bit b chosen by the challenger $\mathcal{C}_{\mathsf{TFHE}}$ is 0, the simulation experiment between $\mathcal{A}_{\mathsf{TFHE}}$ and $\mathcal{Z}$ corresponds exactly to Hyb_6. On the other hand, when the bit b chosen by the challenger $\mathcal{C}_{\mathsf{TFHE}}$ is 1, the experiment between $\mathcal{A}_{\mathsf{TFHE}}$ and $\mathcal{Z}$ corresponds exactly to Hyb_7.

Thus, if the environment $\mathcal{Z}$ can distinguish between the two hybrids with non-negligible probability, then the algorithm $\mathcal{A}_{\mathsf{TFHE}}$ can break the semantic security of the TFHE scheme with non-negligible probability, which is a contradiction.

□

Lemma 13. Hyb_7 *is statistically indistinguishable from* Hyb_8*.*

Proof. The only difference between the two hybrids is that when a corrupt party $\mathcal{P}^*$ initiates the sign-on phase and the ideal functionality $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ responds with ("`SignOn failed`", $\mathsf{sid}, \vec{\mathbf{u}}, \mathsf{msg}, S$), SimHyb proceeds as follows:

– In Hyb_7, SimHyb simulates the one-time encryption of the token on behalf of each honest party $\mathcal{P}_j \in S_0$ as:
$$\mathsf{ct}_j = \mathsf{Token}_j \oplus K_j,$$
where K_j is a uniformly sampled string in $\{0,1\}^\lambda$.

– On the other hand, in Hyb_8, SimHyb simulates the one-time encryption of the token on behalf of each honest party $\mathcal{P}_j \in S_0$ as:
$$\mathsf{ct}_j \leftarrow \{0,1\}^\lambda.$$

Now observe that in both hybrids, when a corrupt party $\mathcal{P}^*$ initiates the sign-on phase and $\mathcal{F}_{\text{FTT}}^{\mathsf{TS}}$ responds with ("`SignOn failed`", $\mathsf{sid}, \vec{\mathbf{u}}, \mathsf{msg}, S$), SimHyb simulates the TFHE encryption of the one-time encryption key K_j as:
$$\mathsf{ct}_{K_j} = \mathsf{TFHE.Enc}(\mathsf{pk}^{\mathsf{TFHE}}, 0^\lambda),$$
which means that in either hybrid, the environment $\mathcal{Z}$ *statistically* has no information about K_j other than via ct_j. Since K_j is a uniformly sampled string in $\{0,1\}^\lambda$, it immediately follows that the distributions of ct_j in hybrids Hyb_7 and Hyb_8 are statistically indistinguishable.

□

B.3 Proof of Theorem 5

Consider an environment $\mathcal{Z}$ who corrupts a party $\mathcal{P}_{\mathcal{A}}$. The strategy of the simulator Sim for our protocol π^{CS} against the environment $\mathcal{Z}$ is described below. Let $\mathsf{Sim.Garble}$ denote the simulator for the garbling scheme. Let $\mathsf{NIZK.Sim}$ denote the simulator of the NIZK argument system and let $\mathsf{NIZK.Ext}$ denote the corresponding extractor. Let $\mathsf{OT.Sim}$ denote the simulator of the OT protocol - let $\mathsf{OT.Sim.Setup}$ denote the algorithm used by the simulator to generate a simulated CRS and $\mathsf{OT.Sim.Round}_2$ denote the algorithm used to generate the second round OT message against a malicious receiver.

B.3.1 Description of Simulator

- **Setup:** First, Sim queries the ideal functionality with "Setup" and the identity of the corrupt party $\mathcal{P}_{\mathcal{A}}$ and receives $(\mathsf{pp}^{\mathsf{TS}}, \mathsf{vk}^{\mathsf{TS}}, \mathsf{sk}^{\mathsf{TS}}_{\mathcal{A}})$ from the ideal functionality. Then, for each $i \in [n]$, Sim does the following:
  - Generate $\mathsf{crs}_{\mathsf{sim}\,i} \leftarrow \mathsf{OT.Sim.Setup}(1^\lambda)$ and a random PRF key (k_i).
  - Give crs_i to $\mathcal{Z}$. If $\mathcal{P}_i \neq \mathcal{P}_{\mathcal{A}}$, also give k_i to $\mathcal{Z}$.
- **Enrollment:** Sim does the following:
  - If an enrollment query was initiated by an honest party, Sim does nothing.
  - If it receives an enrollment query from $\mathcal{Z}$ on behalf of the corrupt party $\mathcal{P}_{\mathcal{A}}$, it forwards it to the ideal functionality.
  - Then, for each enrollment query, for each $i \in [n]$, do the following:
    - Pick $(\overrightarrow{\mathbf{w}}_i, \overrightarrow{\mathbf{v}}_i)$ uniformly at random.
    - Compute $(\mathsf{esk}_i, \mathsf{epk}_i) \leftarrow \mathsf{AHE.Setup}(1^\lambda)$.
    - Let $\overrightarrow{\mathbf{w}}_i = (\mathsf{w}_{i,1}, \ldots, \mathsf{w}_{i,\ell})$. For all $j \in [\ell]$, compute $[\![\mathsf{w}_{i,j}]\!] = \mathsf{AHE.Enc}(\mathsf{epk}_i, \mathsf{w}_{i,j})$.
    - If $\mathcal{P}_i = \mathcal{P}_{\mathcal{A}}$, give $(\overrightarrow{\mathbf{w}}_i, \mathsf{esk}_i, \mathsf{epk}_i, \{[\![\mathsf{w}_{i,j}]\!]\}_{j\in[\ell]})$ to $\mathcal{Z}$. Else, give $(\overrightarrow{\mathbf{v}}_i, \mathsf{epk}_i, \{[\![\mathsf{w}_{i,j}]\!]\}_{j\in[\ell]})$ to $\mathcal{Z}$.

SignOn Phase: Case 1 - Honest Party as $\mathcal{P}_i$
Consider an honest party $\mathcal{P}_i$ that uses an input vector $\overrightarrow{\mathbf{u}}$ and a message msg for which it wants a token by interacting with a set S of two parties, one of which is $\mathcal{P}_{\mathcal{A}}$. Sim gets the tuple (msg, S) from the ideal functionality $\mathcal{F}_{\mathsf{DiFuz}}$ and interacts with the adversary $\mathcal{Z}$ as below:

- $\underline{\textbf{Round 1: } (\mathsf{Sim} \rightarrow)}$ [10] Sim does the following:
  1. For each $j \in [\ell]$, compute the following:
     - Compute ciphertexts $[\![\mathsf{u}_j]\!], [\![\mathsf{u}_j^2]\!], [\![\mathsf{w}_{i,j} \cdot \mathsf{u}_j]\!]$ as encryptions of 0. That is, $[\![\mathsf{u}_j]\!] = \mathsf{AHE.Enc}(\mathsf{epk}_i, 0)$, $[\![\mathsf{u}_j^2]\!] = \mathsf{AHE.Enc}(\mathsf{epk}_i, 0)$ and $[\![\mathsf{w}_{i,j} \cdot \mathsf{u}_j]\!] = \mathsf{AHE.Enc}(\mathsf{epk}_i, 0)$.
     - $\pi_{1,j} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{1,j})$ for the statement $\mathsf{st}_{1,j} = ([\![\mathsf{u}_j]\!], \mathsf{epk}_i) \in L_1$.
     - $\pi_{2,j} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{2,j})$ for the statement $\mathsf{st}_{2,j} = ([\![\mathsf{u}_j]\!], [\![\mathsf{u}_j]\!], [\![\mathsf{u}_j^2]\!], \mathsf{epk}_i) \in L_2$.
     - $\pi_{3,j} \leftarrow \mathsf{NIZK.Sim}(\mathsf{st}_{3,j})$ for the statement $\mathsf{st}_{3,j} = ([\![\mathsf{w}_{i,j}]\!], [\![\mathsf{u}_j]\!], [\![\mathsf{w}_{i,j} \cdot \mathsf{u}_j]\!], \mathsf{epk}_i) \in L_2$.
  2. Send $\mathsf{msg}_1 = (S, \mathsf{msg}, \{[\![\mathsf{u}_j]\!], [\![\mathsf{u}_j^2]\!], [\![\mathsf{w}_{i,j} \cdot \mathsf{u}_j]\!], \pi_{1,j}, \pi_{2,j}, \pi_{3,j}\}_{j\in[\ell]})$ to $\mathcal{A}$.
- $\underline{\textbf{Round 2: } (\rightarrow \mathsf{Sim})}$ On behalf of the corrupt party $\mathcal{P}_{\mathcal{A}}$, receive $([\![\mathsf{x}_2]\!], [\![\mathsf{y}_2]\!], [\![\mathsf{z}_1]\!], [\![\mathsf{z}_2]\!])$ from the adversary.
- $\underline{\textbf{Round 3: } (\mathsf{Sim} \rightarrow)}$ Sim does the following:
  1. Abort if the ciphertexts $([\![\mathsf{x}_2]\!], [\![\mathsf{y}_2]\!], [\![\mathsf{z}_1]\!], [\![\mathsf{z}_2]\!])$ were not correctly computed using the algorithms of AHE, vector $\overrightarrow{\mathbf{v}}_i$ and randomness derived from PRF key k_i.
  2. Generate and send $\mathsf{msg}_3 = \{\mathsf{ot}^{\mathsf{rec}}_{\mathsf{s,t}} = \mathsf{OT.Round}_1(\mathsf{crs}_i, 0)\}_{\mathsf{s}\in\{\mathsf{x,y,z}\},\mathsf{t}\in\{1,2\}}$.

[10]The arrowhead denotes that in this round messages are outgoing from the simulator.

- **Round 4:** $(\rightarrow \mathsf{Sim})$ On behalf of the corrupt party $\mathcal{P}_{\mathcal{A}}$, receive $(\widetilde{\mathcal{C}}, \mathsf{ot}^{\mathsf{sen}}, \mathsf{ct})$ from the adversary.

- **Message to Ideal Functionality $\mathcal{F}_{\mathsf{DiFuz}}$:** Sim does the following:

  1. Abort if $(\widetilde{\mathcal{C}}, \mathsf{ot}^{\mathsf{sen}})$ were not correctly computed using the respective algorithms and randomness derived using PRF key k_i.
  2. Abort if $\mathsf{SKE.Dec}(\mathsf{pad}, \mathsf{ct}) \neq \mathsf{TS.Sign}(\mathsf{sk}^{\mathsf{TS}}_{\mathcal{A}}, \mathsf{msg})$ where pad is derived using PRF key k_i.
  3. Else, instruct the ideal functionality $\mathcal{F}_{\mathsf{DiFuz}}$ to deliver output to the honest party $\mathcal{P}_i$.

SignOn Phase: Case 2 - Malicious Party as $\mathcal{P}_i$
Suppose a malicious party is the initiator $\mathcal{P}_i$. Sim interacts with the adversary $\mathcal{A}$ as below:

- **Round 1:** $(\rightarrow \mathsf{Sim})$ Sim receives $\mathsf{msg}_1 = (S, \mathsf{msg}, \{[\![\mathsf{u}_j]\!], [\![\mathsf{u}_j^2]\!], [\![\mathsf{w}_{i,j} \cdot \mathsf{u}_j]\!], \pi_{1,j}, \pi_{2,j}, \pi_{3,j}\}_{j \in [\ell]})$ from the adversary $\mathcal{A}$ on behalf of two honest parties $\mathcal{P}_j, \mathcal{P}_k$.

- **Round 2:** $(\mathsf{Sim} \rightarrow)$ Sim does the following:

  1. **Message to Ideal Functionality $\mathcal{F}_{\mathsf{DiFuz}}$:**
     1. Run the extractor $\mathsf{NIZK.Ext}$ on the proofs $\{\pi_{1,j}, \pi_{2,j}, \pi_{3,j}\}_{j \in [\ell]}$ compute $\overrightarrow{\mathbf{u}}$.
     2. Query the ideal functionality $\mathcal{F}_{\mathsf{DiFuz}}$ with $\mathsf{inp}_{\mathcal{A}} = (\mathsf{msg}, \overrightarrow{\mathbf{u}}, S)$ to receive output $\mathsf{out}_{\mathcal{A}}$.
  2. Generate $([\![\mathsf{x}_2]\!], [\![\mathsf{y}_2]\!], [\![\mathsf{z}_1]\!], [\![\mathsf{z}_2]\!])$ as encryptions of random messages using public key pk_i and uniform randomness. Send them to $\mathcal{A}$.

- **Round 3:** $(\rightarrow \mathsf{Sim})$ Sim receives $\mathsf{msg}_3 = \{\mathsf{ot}^{\mathsf{rec}}_{\mathsf{s,t}}\}_{\mathsf{s} \in \{\mathsf{x,y,z}\}, \mathsf{t} \in \{1,2\}}$ from the adversary $\mathcal{A}$ on behalf of both honest parties $\mathcal{P}_j$ and $\mathcal{P}_k$.

- **Round 4:** $(\mathsf{Sim} \rightarrow)$ Sim does the following:

  1. Pick a value pad uniformly at random.
  2. if $\mathsf{out}_{\mathcal{A}} \neq \perp$:
     - Let $\mathsf{out}_{\mathcal{A}} = (\mathsf{Token}_j, \mathsf{Token}_k)$.
     - Compute $(\widetilde{\mathcal{C}}_{\mathsf{sim}}, \mathsf{lab}_{\mathsf{sim}}) \leftarrow \mathsf{Sim.Garble}(\mathsf{pad})$.
     - Let $\mathsf{lab}_{\mathsf{sim}} = \{\mathsf{lab}_{\mathsf{s,t}}\}_{\mathsf{s} \in \{\mathsf{x,y,z}\}, \mathsf{t} \in \{0,1\}}$
     - For each $\mathsf{s} \in \{\mathsf{x,y,z}\}$ and each $\mathsf{t} \in \{0,1\}$, compute $\mathsf{ot}^{\mathsf{sen}}_{\mathsf{s,t}} = \mathsf{OT.Sim.Round}_2(\mathsf{lab}_{\mathsf{s,t}})$.
     - Compute $\mathsf{ot}^{\mathsf{sen}} = \{\mathsf{ot}^{\mathsf{sen}}_{\mathsf{s,t}}\}_{\mathsf{s} \in \{\mathsf{x,y,z}\}, \mathsf{t} \in \{0,1\}}$.
     - Set $\mathsf{ct}_j = \mathsf{SKE.Enc}(\mathsf{pad}, \mathsf{Token}_j)$ and $\mathsf{ct}_k = \mathsf{SKE.Enc}(\mathsf{pad}, \mathsf{Token}_k)$.
  3. if $\mathsf{out}_{\mathcal{A}} = \perp$:
     - Compute $(\widetilde{\mathcal{C}}_{\mathsf{sim}}, \mathsf{lab}_{\mathsf{sim}}) \leftarrow \mathsf{Sim.Garble}(\perp)$.
     - Let $\mathsf{lab}_{\mathsf{sim}} = \{\mathsf{lab}_{\mathsf{s,t}}\}_{\mathsf{s} \in \{\mathsf{x,y,z}\}, \mathsf{t} \in \{0,1\}}$
     - For each $\mathsf{s} \in \{\mathsf{x,y,z}\}$ and each $\mathsf{t} \in \{0,1\}$, compute $\mathsf{ot}^{\mathsf{sen}}_{\mathsf{s,t}} = \mathsf{OT.Sim.Round}_2(\mathsf{lab}_{\mathsf{s,t}})$.
     - Compute $\mathsf{ot}^{\mathsf{sen}} = \{\mathsf{ot}^{\mathsf{sen}}_{\mathsf{s,t}}\}_{\mathsf{s} \in \{\mathsf{x,y,z}\}, \mathsf{t} \in \{0,1\}}$.
     - Set $\mathsf{ct}_j = \mathsf{SKE.Enc}(\mathsf{pad}, \mathsf{r}_j)$ and $\mathsf{ct}_k = \mathsf{SKE.Enc}(\mathsf{pad}, \mathsf{r}_k)$ where r_j and r_k are picked uniformly at random.
  4. Send $(\widetilde{\mathcal{C}}_{\mathsf{sim}}, \mathsf{ot}^{\mathsf{sen}}, \mathsf{ct}_j)$ and $(\mathsf{RO}(\widetilde{\mathcal{C}}_{\mathsf{sim}}, \mathsf{ot}^{\mathsf{sen}}), \mathsf{ct}_k)$ to $\mathcal{A}$.

B.3.2 Hybrids

We now show that the above simulation strategy is successful against all environments $\mathcal{Z}$. That is, the view of the corrupt parties along with the output of the honest parties is computationally indistinguishable in the real and ideal worlds. We will show this via a series of computationally indistinguishable hybrids where the first hybrid Hyb_0 corresponds to the real world and the last hybrid Hyb_{11} corresponds to the ideal world.

1. Hyb_0 **- Real World:** In this hybrid, consider a simulator SimHyb that plays the role of the honest parties as in the real world.

2. Hyb_1 **- Remove PRF** In this hybrid, SimHyb, on behalf of every honest party, uses uniformly random strings instead of the output of the pseudorandom function throughout the protocol.

   When Honest Party is $\mathcal{P}_i$:

3. Hyb_2 **- Case 1: Aborts and Message to Ideal Functionality.** In this hybrid, SimHyb aborts if the adversary's messages were not generated in a manner consistent with the randomness output in the setup phase and also runs the query to the ideal functionality.

   That is, SimHyb runs the "$\mathsf{Message\ To\ Ideal\ Functionality}$" step as done by Sim after round 4 of Case 1 of the simulation strategy. SimHyb also performs the Abort check step in step 1 of round 3 of Case 1 of the simulation.

4. Hyb_3 **- Case 1: Simulate NIZKs.** In this hybrid, SimHyb computes simulated NIZK arguments in round 1 of Case 1 as done by Sim in the ideal world.

5. Hyb_4 **- Case 1: Switch Ciphertexts.** In this hybrid, SimHyb computes the ciphertexts in round 1 of Case 1 using random messages as done in the ideal world.

6. Hyb_5 **- Case 1: Switch OT Receiver Messages.** In this hybrid, SimHyb computes the OT receiver messages in round 3 of Case 1 using random inputs as done in the ideal world.

   When Corrupt Party is $\mathcal{P}_i$:

7. Hyb_6 **- Case 2: Message to Ideal Functionality.** In this hybrid, SimHyb runs the "$\mathsf{Message\ To\ Ideal\ Functionality}$" step as done by Sim in round 2 of Case 2 of the simulation strategy. That is, SimHyb queries the ideal functionality using the output of the extractor $\mathsf{NIZK.Ext}$ on the proofs given by $\mathcal{A}$ in round 1.

8. Hyb_7 **- Case 2: Simulate OT Sender Messages.** In this hybrid, SimHyb computes the CRS during the setup phase and the OT sender messages in round 4 of Case 2 using the simulator $\mathsf{OT.Sim}$ as done in the ideal world.

9. Hyb_8 **- Case 2: Simulate Garbled Circuit.** In this hybrid, SimHyb computes the garbled circuit and associated labels in round 4 of Case 2 using the simulator $\mathsf{Sim.Garble}$ as done in the ideal world.

10. Hyb_9 **- Case 2: Switch Ciphertexts.** In this hybrid, SimHyb computes the ciphertexts in round 2 of Case 2 using random messages as done in the ideal world.

11. Hyb_{10} **- Case 2: Switch Ciphertexts.** In this hybrid, in round 4 of the simulation, SimHyb computes the ciphertexts ct_j as done in the ideal world. In particular, if the value $\mathsf{out}_{\mathcal{A}} = \bot$, SimHyb computes ct_j as encryption of a uniformly random value.

12. Hyb_{11}**: Random Shares.** In this hybrid, in both cases, whether honest party or malicious party is the initiator, SimHyb runs the enrollment phase as in the ideal world. That is, each pair $(\vec{\mathbf{w}}_i, \vec{\mathbf{v}}_i)$ is picked uniformly at random instead of as output of the secret sharing scheme. This hybrid corresponds to the ideal world.

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

Lemma 14. Hyb_0 *is statistically indistinguishable from* Hyb_1*.*

Proof. When an honest party initiates the protocol as the querying party $\mathcal{P}_i$, let's say it interacts with parties $\mathcal{P}_j$ and $\mathcal{P}_k$ such that $\mathcal{P}_j$ is corrupt. In Hyb_0, on behalf of $\mathcal{P}_i$, SimHyb checks that the messages sent by both parties $\mathcal{P}_j$ and $\mathcal{P}_k$ are same and if so, computes the output on behalf of the honest party. Since $\mathcal{P}_k$ is honest, this means that if the messages sent by both parties are indeed the same, the adversary $\mathcal{A}$, on behalf of $\mathcal{P}_j$, did generate those messages honestly using the shared randomness generated in the setup phase and the shared values generated in the registration phase.

In Hyb_1, on behalf of $\mathcal{P}_i$, SimHyb checks that the messages sent by the adversary on behalf of $\mathcal{P}_j$ were correctly generated using the shared randomness and shared values generated in the setup and registration phases and if so, asks the ideal functionality to deliver output to the honest party. Thus, the switch from Hyb_0 to Hyb_1 is essentially only a syntactic change. □

Lemma 15. *Assuming the zero knowledge property of the NIZK argument system,* Hyb_1 *is computationally indistinguishable from* Hyb_2*.*

Proof. The only difference between the two hybrids is that in Hyb_1, SimHyb computes the messages of the NIZK argument system by running the honest prover algorithm $\mathsf{Prove}(\cdot)$, while in Hyb_2, they are computed by running the simulator $\mathsf{NIZK.Sim}(\cdot)$. Thus, we can show that if there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{NIZK}}$ that can distinguish between real and simulated arguments with non-negligible probability thus breaking the zero knowledge property of the NIZK argument system which is a contradiction. □

Lemma 16. *Assuming the security of the pseudorandom function,* Hyb_1 *is computationally indistinguishable from* Hyb_2*.*

Proof. The only difference between the two hybrids is that in Hyb_1, SimHyb runs the actual PRF algorithm to compute the randomness for OT, encryption and the garbled circuit whereas, in Hyb_2, SimHyb uses uniformly random strings instead. Observe that since the proofs are already simulated, these PRF keys are not used anywhere else in the protocol. Thus, it is easy to see that if there exists an environment that can distinguish between these two hybrids, we can build an adversary $\mathcal{A}$ that breaks the security of the pseudorandom function which is a contradiction. □

Lemma 17. *Assuming the semantic security of the additively homomorphic encryption scheme* AHE*,* Hyb_3 *is computationally indistinguishable from* Hyb_4*.*

Proof. The only difference between the two hybrids is that in Hyb_3, SimHyb computes the ciphertexts in round 1 by encrypting the honest party's actual inputs $(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{w}}_i)$, while in Hyb_4, the ciphertexts encrypt random messages. Thus, we can show that if there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{AHE}}$ that can distinguish between encryptions of the honest party's actual inputs and encryptions of random messages with non-negligible probability thus breaking the semantic security of the encryption scheme AHE which is a contradiction. □

Lemma 18. *Assuming the security of the oblivious transfer protocol* OT *against a malicious sender,* Hyb_4 *is computationally indistinguishable from* Hyb_5*.*

Proof. The only difference between the two hybrids is that in Hyb_4, SimHyb computes the OT receiver's messages as done by the honest party in the real world, while in Hyb_5, the OT receiver's messages are computed by using random messages as input. Thus, we can show that if there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{OT}}$ that can distinguish between OT receiver's messages of the honest party's actual inputs and random inputs with non-negligible probability thus breaking the security of the oblivious transfer protocol OT against a malicious sender which is a contradiction. □

Lemma 19. *Assuming the argument of knowledge property of the NIZK argument system,* Hyb_5 *is computationally indistinguishable from* Hyb_6*.*

Proof. The only difference between the two hybrids is that in Hyb_6, SimHyb also runs the extractor $\mathsf{NIZK.Ext}$ on the proofs given y the adversary to compute its input $\overrightarrow{\mathbf{u}}$. Thus, the only difference between the two hybrids is if the adversary can produce a set of proofs $\{\pi_j\}$ such that, with non-negligible probability, all of the proofs verify successfully, but SimHyb fails to extract $\overrightarrow{\mathbf{u}}$ and hence SimHyb aborts.

However, we can show that if there exists an environment $\mathcal{Z}$ that can this to happen with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{NIZK}}$ that breaks the argument of knowledge property of the system NIZK with non-negligible probability which is a contradiction. □

Lemma 20. *Assuming the security of the oblivious transfer protocol* OT *against a malicious receiver,* Hyb_6 *is computationally indistinguishable from* Hyb_7*.*

Proof. The only difference between the two hybrids is that in Hyb_6, SimHyb computes the OT sender's messages by using the actual labels of the garbled circuit as done by the honest party in the real world, while in Hyb_7, the OT sender's messages are computed by running the simulator $\mathsf{OT.Sim}$. In Hyb_7, the crs in the setup phase is also computed using the simulator $\mathsf{OT.Sim}$.

Thus, we can show that if there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{OT}}$ that can distinguish between the case where the crs an OT sender's messages were generated by running the honest sender algorithm from the case where the crs and the OT sender's messages were generated using the simulator $\mathsf{OT.Sim}$ with non-negligible probability thus breaking the security of the oblivious transfer protocol OT against a malicious receiver which is a contradiction. $\square$

Lemma 21. *Assuming the correctness of the extractor* $\mathsf{NIZK.Ext}$ *and the security of the garbling scheme,* Hyb_7 *is computationally indistinguishable from* Hyb_8.

Proof. The only difference between the two hybrids is that in Hyb_7, SimHyb computes the garbled circuit by running the honest garbling algorithm Garble using honestly generated labels, while in Hyb_8, SimHyb computes a simulated garbled circuit and simulated labels by running the simulator $\mathsf{Sim.Garble}$ on the value $\mathsf{out}_{\mathcal{A}}$ output by the ideal functionality. From the correctness of the extractor $\mathsf{NIZK.Ext}$, we know that the output of the garbled circuit received by the evaluator $\mathcal{Z}$ in Hyb_7 is identical to the output of the ideal functionality $\mathsf{out}_{\mathcal{A}}$ used in the ideal world. Thus, we can show that if there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{Garble}}$ that can distinguish between an honestly generated set of input wire labels and an honestly generated garbled circuit from simulated ones with non-negligible probability thus breaking the security of the garbling scheme which is a contradiction. $\square$

Lemma 22. *Assuming the circuit privacy property of the additively homomorphic encryption scheme* AHE*,* Hyb_8 *is computationally indistinguishable from* Hyb_9.

Proof. The only difference between the two hybrids is that in Hyb_8, SimHyb computes the ciphertexts sent in round 2 by performing the homomorphic operations on the adversary's well-formed ciphertexts sent in round 1 exactly as in the real world, while in Hyb_9, SimHyb generates ciphertexts that encrypt random messages. Thus, we can show that if there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability, we can design a reduction $\mathcal{A}_{\mathsf{AHE}}$ that can break the circuit privacy of the circuit private additively homomorphic encryption scheme AHE which is a contradiction. $\square$

Lemma 23. *Assuming the semantic security of the secret-key encryption scheme* SKE*,* Hyb_9 *is computationally indistinguishable from* Hyb_{10}.

Proof. The only difference between the two hybrids is in the scenario when the malicious party initiates the session, and in round 4, the value $\mathsf{out}_{\mathcal{A}}$ received from the ideal functionality is $\perp$. In Hyb_9, the simulator computes the ciphertext ct_j as an encryption of the token Token_j as done by the honest party in the real world whereas in Hyb_{10}, the simulator computes the ciphertext ct_j in this case as an encryption of a random string. Observe that the secret key used for encrypting is not used anywhere else in the experiment. Thus, it is easy to see that if there exists an environment $\mathcal{Z}$ that can distinguish between the two hybrids with non-negligible probability, we can use $\mathcal{Z}$ to construct an adversary $\mathcal{A}_{\mathsf{SKE}}$ that breaks the semantic security of the encryption scheme which is a contradiction. $\square$

Lemma 24. *Assuming the security of the secret sharing scheme,* Hyb_{10} *is computationally indistinguishable from* Hyb_{11}.

Proof. The only difference between the two hybrids is that in Hyb_{10}, for every enrollment with a template $\overrightarrow{\mathbf{w}}$, for each $i \in [n]$, the corrupt party $\mathcal{P}_i$ gets a vector $\overrightarrow{\mathbf{w}}_i$ or $\overrightarrow{\mathbf{v}}_i$ that is picked as a share of $\overrightarrow{\mathbf{w}}$ by running the secret sharing algorithm Share whereas, in Hyb_{11}, these values are picked uniformly at random by the simulator. Since the threshold used in the secret sharing scheme is two and no information about the honest parties' share is even known to the simulator, it is easy to see that if there exists an environment that can distinguish between these two hybrids, we can build an adversary $\mathcal{A}$ that breaks the security of the threshold secret sharing scheme which is a contradiction. $\square$