Efficient Privacy-Preserving Blueprints for Threshold Comparison

Pratyush Ranjan Tiwari[1], Harry Eldridge[2], and Matthew Green[2]

[1] Eternis Labs
pratyush@eternis.ai
[2] Johns Hopkins University
{hme, mgreen}@cs.jhu.edu

Abstract. Privacy-Preserving Blueprints (PPBs), introduced by Kohlweiss *et al.* in in EUROCRYPT 2023, offer a method for balancing user privacy and bad-actor detection in private cryptocurrencies. A PPB scheme allows a user to append a verifiable *escrow* to their transactions which reveals some identifying information to an authority in the case that the user misbehaved. A natural PPB functionality is for escrows to reveal user information if the user sends an amount of currency over a certain threshold. However, prior works constructing PPBs for such a functionality have severe limitations when it comes to efficiency: escrows are either computationally infeasible to compute, or too large to be plausibly stored on a large-scale distributed ledger. We address these gaps by constructing a space and computation-efficient PPB for threshold comparison, producing escrows under 2kb that can be computed in seconds. The scheme can be instantiated using well-known cryptographic primitives, namely variants of the ElGamal encryption scheme and generic non-interactive zero-knowledge proofs. As an additional contribution, we implement one of the theoretical generic PPB constructions originally proposed by Kohlweiss *et al.* and find that it performs surprisingly well in practice. For the threshold comparison functionality it requires approximately 14kb escrows, and can be computed in around 12 seconds.

1 Introduction

Cryptocurrencies, protocols and exchanges have recently faced stringent regulatory pressure due to their potential misuse by terrorist organizations and for money laundering purposes in the US [28], EU [26], UAE [7] and Japan [31]. This problem has become more serious due to the recent deployment of asset-backed *stablecoins*, which use decentralized networks to conduct transactions denominated in regulated financial assets. A common solution to this problem employs blockchain tracing [18], which depends on the fact that many cryptocurrency systems use *non-private* transactions, where all details are visible to the public. Unfortunately, this results in a level of transaction privacy dramatically below that of the traditional financial system. Several technologies have sought to improve privacy by deploying techniques that make transaction details confidential to non-participants [25, 21, 24, 16]. The lack of visibility in these systems has led to a series of conflicts with regulatory authorities, culminating in the U.S. Treasury applying sanctions to the Tornado Cash smart contracts [29].

Over the past several years, several proposals have sought to achieve privacy while also satisfying the requirements of government regulators. One family of solutions aims to preserve the confidentiality of transactions from the general public while also providing some degree of visibility to a designated *compliance authority* (*e.g.,* a stablecoin issuer). The most straightforward approach to this problem makes *all* transaction data visible to the authority, for example by encrypting transaction details under the authority's public key. However, this level of visibility can pose a risk to users in the event that the authority becomes compromised. An alternative approach is related to the area of *pre-constrained encryption* [2]: in these protocols a designated authority learns only appropriate functions of an individual's financial activity, while keeping other aspects fully private. For example, individuals who remain within an envelope of compliance parameters would experience full transaction privacy, while those who exceed these parameters would have some or all of their transaction data revealed. Most critically, this guarantee should hold *even if the authority is fully compromised and behaves maliciously*, which rules out solutions that simply collect all transaction data and then filter what is made available to regulators.

Privacy-preserving blueprints. In service of this goal, Kohlweiss et al. [13] introduced *Privacy-Preserving Blueprints* (PPBs). In a PPB scheme, an auditor publishes a function $f(\cdot,\cdot)$, along with some secret and committed input x chosen by the auditor. Each time a user authors a transaction, it adds an encrypted *escrow* field that contains the value $f(x, v)$, where v represents some input from the user's transaction. Finally, this escrow field can be decrypted by the auditor. This primitive has several applications. For example, $f(x, \cdot)$ can realize a *watchlist* functionality [13, 11]: here $x = \{A_1, \ldots, A_n\}$ contains a secret list of suspicious accounts, and $v = \{A_{\mathsf{sender}}, A_{\mathsf{recipient}}\}$ are the addresses of the parties involved in the transaction. The function $f(x, v)$ would reveal v to the auditor iff $v \cap x \neq \emptyset$ and would reveal an empty value $\perp$ in all other cases. A critical feature of PPBs is that the watchlist x remains confidential from the transaction authors.[3]

PPBs for threshold comparison. One particularly useful functionality for PPBs is *threshold comparison*, which allows authorities to place threshold limits on transaction values. Here v contains the amount of a transaction, and the auditor learns some message m iff $v > x$.[4] This functionality is a critical component of implementing traditional Anti-Money Laundering (AML) functionalities. For

[3] In practice, the auditor's input x can be placed within a public commitment: this ensures that the contents of the watchlist are not entirely arbitrary, *e.g.,* outside parties can be allowed to examine the content of the watchlist and ensure that this watchlist is in use on the network.

[4] In practice m may be identifying information for the user, or a key that encrypts other values of interest.

example, the U.S. Bank Secrecy Act both mandates the reporting of cash transactions over $10,000 [19], and that banks file secret Suspicious Activity Reports (SARs) for any transaction that may represent "structuring" intended to evade the published limits. PPBs for threshold comparison (over individual transaction amounts, or cumulative transaction amounts over a period) represents one way to implement SARs on a private chain.

Building threshold-comparison PPBs for use on large-scale distributed ledgers imposes a number of restrictions on constructions. Any constructions must be efficient in both computation time and particularly in bandwidth (*i.e.,* the size of the escrow field), since the escrow must be stored on the ledger. Next, PPBs require that escrows be *publicly verifiable*. While the auditor checks and decrypts values, the validators maintaining the ledger should be capable of verifying that any given escrow is valid as a condition for accepting transactions on chain. Finally, the auditor must be able to decrypt an escrow field non-interactively, *i.e.,* it should not require further interaction with the transaction author to obtain the escrow data.

Limitations of previous work. Previous works addressing threshold-comparison PPBs or related problems fall short on one or more of these metrics. Kohlweiss *et al.* propose a solution based on fully-homomorphic encryption (FHE) [13], which can realize general functionalities and is space-efficient. However, the scheme requires the transactor to formulate a zero-knowledge proof over the FHE evaluation process, which remains computationally infeasible [30]. Subsequent work [11] improves efficiency, but supports only the more limited set membership ("watchlist") functionality discussed above.

A second line of work by David *et al.* [8] produces a computationally-efficient *updatable* PPB that can be adapted to fit the threshold comparison functionality. In their scheme, users submit transactions with values up to some limit d, and the auditor becomes able to decrypt an escrow when the sum of the transaction values passes the threshold t. Unfortunately, their solution is notably inefficient when it comes to the escrow size: escrows require a number of group elements that scales *linearly* with d. As a concrete example, to allow up to 1000 units of currency per transaction would require each escrow to consume 385KB using the instantiation from [8, §8].

The PPB offering the most practical solution for threshold-comparison is a second construction in Kohlweiss *et al.* [13] which uses non-interactive secure computation (NISC). In this construction, escrows consist of the sender message in a NISC protocol (typically instantiated with oblivious-transfer (OT) ciphertexts and a garbled circuit) along with a proof that the NISC was computed honestly. While computing a "proof of garbling" requires proving over cryptographic primitives, and escrows must contain the full garbled truth tables, this solution remains plausible as a deployable PPB, though to the best of the authors' knowledge no implementations currently exist. One contribution of this work is to explore and optimize this approach.

Table 1: A comparison of escrow sizes and proof generation time for threshold comparison of 32-bit integers. Best-case FHE escrow size was estimated as TFHE [6] ciphertexts compressed using the TFHE-rs library [22]. The d in the escrow size for [8] scales linearly with the amount of currency permitted per-transaction, and would be impractically large for implementations at $d = 2^{32}$ in this example.

Scheme	**Practical?**	**Generation time**	**Estimated Escrow Size (bytes)**
Kohlweiss *et al.* [13] FHE	No	n/a	2,252
David *et al.* [8]	No	n/a	$384d + 1{,}152$
Kohlweiss *et al.* [13] NISC	Yes	11.95 seconds (see §5.)	13,472
This work	Yes	4.42 seconds	1,600

Our contributions. In this work we investigate the problem of constructing concretely efficient privacy-preserving blueprints for threshold comparison. We investigate two different approaches to this question:

1. **Realizing and optimizing NISC for PPBs.** As a baseline, we revisit the NISC approach mentioned by Kohlweiss *et al.* [13]. While the authors describe this construction as "theoretical" for general functionalities, we show that careful engineering allows us to realize the specific comparison functionality practically. The challenge here is careful circuit optimization, as well as the need to realize a ZK-friendly implementation of a garbled circuit, something that requires non-trivial optimization and design choices. We use this construction as a baseline to compare against later contributions.
2. **Developing a novel PPB based on ambiguous encryption.** To improve upon this result, we next develop a novel PPB construction that allows the auditor to perform an oblivious variant of the classic string comparison algorithm using "lossy" and ambiguous encryption [3]. Our construction makes use of the fact that, in our intended application, the threshold comparison functionality already leaks information about the user's input amount *in the specific case where the comparison is satisfied.* The resulting escrows require only $4\log_b(\ell) + 7$ group elements, where ℓ is the maximum value of the threshold t and b is an arbitrary base chosen at setup time.

To verify the efficiency of our techniques we implement both approaches under the assumption of a malicious user, and we present detailed performance and bandwidth numbers for each construction. We provide a comparison of estimated escrow sizes and prover times between this and prior work in Table 1.
In the next section we provide a high-level technical overview of our contributions.

1.1 Technical Overview

We are interested in a PPB for the following functionality:

$$f(t,(v,m)) = \begin{cases} m \text{ if } v > t \\ \perp \text{ otherwise} \end{cases}$$

In other words, our goal is to construct an encryption scheme where a ciphertext ct can only be decrypted if some associated value v is greater than a private threshold t, which is encoded in the public key.

Exploring and optimizing the NISC-based construction Kohlweiss *et al.* observe that a "generic" construction of PPBs can be derived from the use of non-interactive secure computation (NISC) and an appropriate ZK proof-of-knowledge [13]. However, they do not implement this construction, and mainly treat it as a "theoretical" alternative to their direct construction of a different functionality. Thus, as a baseline we wish to determine if this protocol can work in practice for the specific threshold comparison functionality we are interested in.

In practice, realizing this construction requires solving a number of engineering challenges. NISC constructions traditionally involve the use of a garbled circuit combined with a re-usable two-message OT scheme. The first challenge, therefore, is to identify an efficient circuit for threshold comparison that maximizes the advantages of efficient garbling schemes, such as the ability to calculate XOR gates for "free" [15]. A more imposing challenge is the need to construct efficient zero-knowledge proofs ensuring that the garbled circuit has been constructed correctly. We accomplish this using efficient zkSNARKs [12]. We show that even complex garbling optimizations such as Free-XOR [15] and Half-Gates [34] can be realized with careful optimizations, by storing wire labels and circuit tables as field elements, and by using zero-knowledge-friendly hash functions such as Poseidon [10].[5] While we use this result mainly as a baseline for comparing our novel scheme below, we find that the resulting implementation is borderline practical. As shown in Table 1, for 32-bit comparisons we are able to realize our escrows in less than 14KB, with approximately 12 seconds of proving time.

Our Scheme Garbled circuits by definition hide the full state of the computation up to the circuit's output. In the context of a PPB for threshold comparison, this means that the state of the comparison between t and v must remain completely hidden, even though m may *fully reveal* v, along with other information about the user. Our scheme improves upon the baseline by allowing information about v to be leaked in the case that $v > t$. Specifically, we build a PPB for the following functionality:

$$f_{thr}(t, (v, m)) = \begin{cases} (m, f'_b(v, t)) \text{ if } v > t \\ \perp \text{ otherwise} \end{cases}$$

[5] A surprising difficulty in this work is that the free-XOR technique cannot be natively employed on field elements, since the field addition operation is not its own inverse. The practical impact is that we must frequently convert lables between bit-string and field-element representation in our zk circuit, adding to the prover's computational overhead.

where f_b' represents some extra information about v in relation to t. We will fully describe f_b' after going over the basics of our construction.

We now describe our main construction. Let ℓ be the maximum possible value of t. We can construct a PPB for f_{thr} as follows:

- The auditor publishes a public key consisting of $\ell + 1$ public keys for an underlying encryption scheme: $pk_0, \ldots, pk_\ell$, the first t of which are *lossy*, i.e. cannot decrypt any messages encrypted to them.
- The escrow is computed as $ct \leftarrow \mathsf{Enc}(pk_v, m)$, along with a zero-knowledge proof that ct was computed correctly.

If $v \leq t$, then pk_v is lossy, and so ct cannot be decrypted. Correspondingly, if $v > t$ then pk_v is non-lossy, and the message can be recovered. To decrypt the receiver tries all of their non-lossy secret keys until a decryption succeeds.

Unfortunately, this construction requires an impractically large public key that scales linearly with the maximum threshold. To solve this issue, we alter our construction to allow for a tradeoff between public key and ciphertext size. We achieve this tradeoff by using multiple copies of the above construction, where each copy, or "row," represents one digit of t in some base b. For example, if we are using base 10, max threshold $\ell = 99$, and $t = 42$, then the public key would consist of two rows of keys: $pk_0^0, \ldots, pk_0^{10}$ and $pk_1^0, \ldots, pk_1^{10}$, where $\{pk_0^0, pk_0^1, pk_0^2, pk_0^3, pk_0^4\}$ and $\{pk_1^0, pk_1^1, pk_1^2\}$ are lossy. Going forward, let $t[i]$ denote the digit of t at position i.

Say our value $v = 48$. To encrypt, we can start by computing the value $ct_0^{\mathsf{R}} \leftarrow \mathsf{Enc}(pk_4^0, m)$, as if $t[0] < 4$, then v is clearly larger than t. But what if $t[0] = 4$? In this case, we would like decryption to be possible if and only if $t[0] = 4$ *and* $t[1] < 8$. To achieve this, we can sample a one-time-pad (OTP) α, and compute the following two ciphertexts:

- $ct_0^{\mathsf{M}} \leftarrow \mathsf{Enc}(pk_0^5, \alpha)$
- $ct_1^{\mathsf{R}} \leftarrow \alpha \circ \mathsf{Enc}(pk_1^8, m)$

where $\alpha \circ x$ denotes blinding x with the one-time-pad. The final ciphertext is $ct = (ct_0^{\mathsf{R}}, ct_0^{\mathsf{M}}, ct_1^{\mathsf{R}})$. To decrypt, the receiver first attempts to decrypt ct_0^{R} with each non-lossy sk_0^i. If none of these succeed, they compute $ct_1^{\mathsf{R}'} \leftarrow \mathsf{Dec}(sk_0^5, ct_0^{\mathsf{M}}) \circ ct_1^{\mathsf{R}}$, attempt to decrypt $ct_1^{\mathsf{R}'}$ with each non-lossy sk_1^i, and if none of those succeed output $\perp$. We refer to ct_i^M as a "match" ciphertext, as it needs to be decrypted when a digit of v exactly equals a digit of t, and ct_i^R as a "reveal" ciphertext, as it directly reveals the message.

We note here the conflicting requirements between the match and reveal ciphertexts. Reveal ciphertexts must indicate whether a decryption succeeds in order for the receiver to know that they have successfully decrypted the message, i.e. they must be *robust*. However, match ciphertexts must *hide* whether decryption was successful, as revealing so would also reveal information about the value v, even if $v < t$. Practically, this means that we must use different encryption schemes for the match and reveal ciphertexts. Additionally, both schemes must satisfy some form of *ambiguity*, which ensures that the ciphertexts don't reveal what public keys they were encrypted under.

The above technique can be extended to thresholds of any number of digits. To do so, a match ciphertext after the first encrypts not just a new one-time pad α_i, but instead $\alpha_i \circ \alpha_{i-1}$, i.e. a pad blinded by the *previous* OTP. This ensures that a reveal ciphertext can only be decrypted if *all* previous digits have been matched.

We can now see what $f'_b(v,t)$ reveals about v. The auditor learns the longest (base b) prefix of v such that each digit is greater than or equal to the corresponding digit of t.

2 Preliminaries

We use $[x,y]$ to denote the set $\{x, x+1, \ldots, y\}$, where $x, y \in \mathbb{Z}$ and $x \leq y$. We use $[x]$ as shorthand to denote $[1,x]$. When $x > y$ we allow $[x,y]$ to denote $\{x, x-1, \ldots, y\}$. For an integer x we use $x[i]_b$ to denote the 1-indexed i'th digit of x in base b, e.g. for the base-10 value $x = 357$, $x[1]_{10} = 3$. We may drop the subscript when the base is clear from context. We denote concatenation between the strings x and y with $x||y$, and the empty string as ε. For any algorithm $\mathcal{A}$, we use $y := \mathcal{A}(x;r)$ to denote that the algorithm runs on input x and random tape r. In most cases, we do not denote the random tape explicitly and simply write $y \leftarrow \mathcal{A}(x)$. We may write $y \overset{r}{\leftarrow} \mathcal{A}(x)$ to indicate sampling r as appropriate and recording it for later use. We use λ as the computational security parameter, and $\mathsf{negl}(\lambda)$ to denote negligible functions. For two probability ensembles $\{A_i\}_i$ and $\{B_i\}_i$, we use $\{A_i\}_i \overset{\mathrm{p}}{=} \{B_i\}_i$ to denote that the ensembles are identical, $\{A_i\}_i \overset{\mathrm{s}}{\approx} \{B_i\}_i$ to denote that the ensembles are statistically close, and $\{A_i\}_i \overset{\mathrm{c}}{\approx} \{B_i\}_i$ to denote that the ensembles are computationally indistinguishable. We use $\Pr[E : A]$ to denote the probability of an event E in an experiment defined by executing A. For a finite set $\mathcal{X}$, we will use $\mathcal{U}_{\mathcal{X}}$ to denote the uniform distribution over $\mathcal{X}$. For a distribution $\mathcal{D}$ we will use $x \leftarrow \mathcal{D}$ to indicate sampling a value according to the distribution. We will use f_{thr} to denote the following function:

$$f_{thr}(t,(v,m)) = \begin{cases} (m, f'_b(v,t)) \text{ if } v > t \\ \perp \text{ otherwise} \end{cases}$$

Where $f'_b(v,t)$ is defined as follows.

$f'_b(v,t)$

$x = \varepsilon$
for $i \in [\lceil \log_b(t) \rceil]$
  if $v[i]_b \geq t[i]_b$ **then** $x = x||v[i]_b$ **else break**
return x

We defer descriptions of the Diffie-Hellman assumption, the ElGamal encryption scheme, one-time pads, and non-interactive zero-knowledge proofs to the full version of the paper.

2.1 Lossy Public-Key Encryption

Definition 1. *A lossy public-key encryption scheme (PKE) [23] for a message space* $\mathcal{M}$ *is a tuple of algorithms* $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{KGen}_{\mathsf{loss}}, \mathsf{Enc}, \mathsf{Dec})$ *defined as:*

- $\mathsf{Setup}(1^\lambda) \rightarrow \Lambda_k$*: outputs public parameters* Λ_k*.*
- $\mathsf{KGen}(\Lambda_k) \rightarrow (sk, pk)$*: generates a keypair* (sk, pk)*.*
- $\mathsf{KGen}_{\mathsf{loss}}(\Lambda_k) \rightarrow pk$*: generates a lossy key* pk*.*
- $\mathsf{Enc}(\Lambda_k, pk, m) \rightarrow ct$*: takes as input a public key* pk *and a plaintext message* $m \in \mathcal{M}$ *and outputs a ciphertext* ct*.*
- $\mathsf{Dec}(\Lambda_k, sk, ct) \rightarrow m$*: takes as input a secret key* sk *and a ciphertext* ct *and either outputs* $\perp$ *or the message* m*.*

We informally describe lossiness along with some additional properties we will require of our PKE scheme here. When invoking the PKE algorithms we may drop the Λ_k input for brevity when the value is clear from context. We defer the definitions of standard PKE properties, as well as the formalized versions of the properties below, to the full version of the paper.

Lossiness. We require that messages encrypted to keys generated by $\mathsf{KGen}_{\mathsf{loss}}$ are unrecoverable, even by the party that created the keys, and refer to such keys as *lossy*. We refer to non-lossy keys as *injective*. We additionally require that adversaries should be unable to distinguish between lossy and injective keys.

Robustness and Collision Freeness. We will require that the encryption schemes used in this work satisfy properties related to their behavior when a ciphertext is decrypted with the *wrong* key (i.e. a secret key unrelated to the public key the ciphertext was encrypted under). A *robust* encryption scheme [1, 20] requires that decrypting with the wrong key results in the output $\perp$. The weaker *collision-free* [20] requirement stipulates that decrypting an encryption of some message m with the wrong key results in an output $m' \neq m$.

Lossy Ambiguity. An *ambiguous* PKE [3] requires that a decryptor be unable to tell which key a ciphertext was encrypted under. We will require two variants of this definition in relation to *lossy* keys. *Weak* lossy ambiguity will require ambiguity between ciphertexts encrypted under lossy keys, while the *strong* variant will require that ambiguity still hold even between lossy and injective keys.

Lossy Correlation Resistance. In order to preserve soundness, we will need a client to be unable to cause the PPB decryption process to terminate early. This in turn means that they should be unable to craft lossy ciphertexts such that their decryptions appear to be valid ciphertexts for some non-lossy key. We refer to this property as *lossy correlation resistance*.

2.2 Statistically Hiding Commitments

Definition 2. *A statistically hiding non-interactive commitment scheme is a tuple of algorithms* $\mathsf{C} = (\mathsf{CSetup}, \mathsf{Commit})$ *defined over a message space* $\mathbb{M}$ *and randomness space* $\mathbb{R}$ *defined as:*

- $\mathsf{CSetup}(1^\lambda) \to \Lambda_C$*: outputs public parameters* Λ_C*.*
- $\mathsf{Commit}(\Lambda_C, m; r) \to C$*: takes as input a message* $m \in \mathbb{M}$ *and some randomness* $r \in \mathbb{R}$ *and outputs a commitment* C*.*

We require that the above algorithms satisfy the following properties.

- ***Statistical Hiding****: For any* Λ_C *output by* $\mathsf{CSetup}(1^\lambda)$ *and for any* $m_0, m_1 \in \mathbb{M}$ *is should hold that*

$$\{\mathsf{Commit}(\Lambda_C, m_0; r) \,:\, r \leftarrow \mathbb{R}\} \overset{s}{\approx} \{\mathsf{Commit}(\Lambda_C, m_1; r) \,:\, r \leftarrow \mathbb{R}\}$$

- ***Computational Binding****: For all PPT adversaries* $\mathcal{A}$ *is should hold that*

$$\Pr\left[\begin{matrix}\mathsf{Commit}(\Lambda_C, m_0; r_0) = \mathsf{Commit}(\Lambda_C, m_1, r_1) \\ \wedge\, m_0 \neq m_1\end{matrix} : \begin{matrix}\Lambda_C \leftarrow \mathsf{CSetup}(1^\lambda) \\ (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(\Lambda_C)\end{matrix}\right] \leq \mathsf{negl}(\lambda)$$

2.3 Privacy Preserving Blueprints

We reproduce here the PPB definition put forth in [8], adapted to the non-updatable case.

Definition 3. *Let* $\mathsf{C} = (\mathsf{CSetup}, \mathsf{Commit})$ *be a statistically hiding non-interactive commitment scheme defined over* $(\mathbb{M}, \mathbb{R})$*, and* $f : \mathbb{M} \times \mathbb{M} \to \mathbb{M}$ *be a function. A Privacy Preserving Blueprint (f-PPB) scheme is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{VerPK}, \mathsf{Escrow}, \mathsf{VerEscrow}, \mathsf{Dec})$ *defined as:*

- $\mathsf{Setup}(1^\lambda, \Lambda_C) \to (\Lambda, \mathsf{td})$*: takes as input the security parameter and the public parameters of the commitment scheme, and outputs parameters for the scheme* Λ*, which will contain* Λ_C*, along with a trapdoor* td*.*
- $\mathsf{KGen}(\Lambda, t, r) \to (sk, pk)$*: takes as input values* $t \in \mathbb{M}$ *and* $r \in \mathbb{R}$ *and outputs a key pair* (sk, pk)*.* (t, r) *define a commitment* C_{KG}*.*
- $\mathsf{VerPK}(\Lambda, pk, C_{KG}) \to b$*: takes as input a public key and a commitment, and outputs a bit* $b \in \{0, 1\}$ *verifying the validity of* pk*.*
- $\mathsf{Escrow}(\Lambda, pk, m, r) \to Z/\bot$*: takes as input a public key, a message, and some randomness, and returns an escrow value* Z *for the commitment* C_E *defined by* (m, r)*, or* $\bot$*.*
- $\mathsf{VerEscrow}(\Lambda, pk, C_E, Z) \to b$*: takes as input a public key, a commitment, and an escrow, and outputs a bit* $b \in \{0, 1\}$ *verifying the validity of the escrow.*
- $\mathsf{Dec}(\Lambda, sk, C_E, Z) \to m'/\bot$*: takes as input a secret key, a commitment, and an escrow, and returns* $m' = f(t, m) \in \mathbb{M}$ *or* $\bot$*.*

We require that a PPB satisfy the following properties:

- ***Correctness**: For all* $\Lambda_C \leftarrow \mathsf{CSetup}(1^\lambda)$, $\Lambda \leftarrow \mathsf{Setup}(1^\lambda, \Lambda_C)$, *all* $t, m \in \mathbb{M}$, *and all* $r \in \mathbb{R}$*:*

$$\Pr\left[\mathsf{Dec}(\Lambda, sk, C, Z) \neq f(t,m) \;:\; \begin{array}{r} (sk, pk) \leftarrow \mathsf{KGen}(\Lambda, t) \\ Z \leftarrow \mathsf{Escrow}(\Lambda, pk, m, r) \\ C \leftarrow \mathsf{Commit}(\Lambda_C, m; r) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

- ***Soundness**: For all PPT adversaries* $\mathcal{A}$*, the adversary's advantage in the following game is negligible:*

$\underline{\mathsf{Sound}(\mathsf{PPB}, \mathcal{A}, \lambda)}$

$\Lambda_C \leftarrow \mathsf{CSetup}(1^\lambda)$

$(\Lambda, \cdot) \leftarrow \mathsf{Setup}(1^\lambda, \Lambda_C)$

$t, r_k \leftarrow \mathcal{A}(1^\lambda, \Lambda)$

$(pk, sk) \leftarrow \mathsf{KGen}(\Lambda, t, r_k)$

$(C_E, m, r_E, Z) \leftarrow \mathcal{A}(pk)$

return $[C_E = \mathsf{Commit}(\Lambda_C, m; r_E) \wedge$

$\quad \mathsf{VerEscrow}(\Lambda, pk, C_E, Z) \wedge \mathsf{Dec}(\Lambda, sk, C_E, Z) \neq f(t,m)]$

The $\mathsf{Sound}(\mathsf{PPB}, \mathcal{A}, \lambda)$ *advantage of* $\mathcal{A}$ *is defined as:*

$$\mathsf{Adv}^{\mathsf{Sound}}_{\mathsf{PPB},\mathcal{A}}(\lambda) = \Pr\left[\mathsf{Sound}(\mathsf{PPB}, \mathcal{A}, \lambda) \Rightarrow 1\right]$$

- ***Threshold Hiding**: For* $b \in \{0,1\}$ *let* $\mathsf{ThrHide}_b(\mathsf{PPB}, \mathcal{A}, \lambda)$ *denote the result of the following probabilistic experiment:*

$\underline{\mathsf{ThrHide}_b(\mathsf{PPB}, \mathcal{A}, \lambda)}$

$\Lambda_C \leftarrow \mathsf{CSetup}(1^\lambda)$

$(\Lambda, \cdot) \leftarrow \mathsf{Setup}(1^\lambda, \Lambda_C)$

$(t_0, t_1) \leftarrow \mathcal{A}(\Lambda)$

$r \leftarrow\!\!\$\; \mathbb{R}$

$(\cdot, pk) \leftarrow \mathsf{KGen}(\Lambda, t_b, r)$

$B \leftarrow \mathcal{A}(pk)$

The $\mathsf{ThrHide}(\mathsf{PPB}, \mathcal{A}, \lambda)$ *advantage of* $\mathcal{A}$ *is defined as:*

$$\mathsf{Adv}^{\mathsf{ThrHide}}_{\mathsf{PPB},\mathcal{A}}(\lambda) = \left|\Pr\left[\mathsf{ThrHide}_0(\mathsf{PPB}, \mathcal{A}, \lambda) \Rightarrow 1\right] - \Pr\left[\mathsf{ThrHide}_1(\mathsf{PPB}, \mathcal{A}, \lambda) \Rightarrow 1\right]\right|$$

A PPB satisfies threshold hiding if for all PPT adversaries $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{ThrHide}}_{\mathsf{PPB},\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

- ***Escrow Hiding**: For* $b \in \{0,1\}$*, let* $\mathsf{EscHide}_b(\mathsf{PPB}, \mathcal{A}, \lambda)$ *denote the result of the following probabilistic experiment with a simulator* Sim*:*

$\underline{\mathsf{EscHide}_b(\mathsf{PPB}, \mathcal{A}, \mathsf{Sim}, \lambda)}$

$\Lambda_C \leftarrow \mathsf{CSetup}(1^\lambda)$
$(\Lambda, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \Lambda_C)$
$(t, r_{KG}, pk, x) \leftarrow \mathcal{A}(\Lambda)$
if $\mathsf{VerPK}(\Lambda, pk, \mathsf{Commit}(\Lambda_C, t; r_{KG})) = 0$: **return** $\perp$
$r \leftarrow\$ \mathbb{R}$
$Z \leftarrow$ **if** $b = 0$ **then** $\mathsf{Escrow}(\Lambda, pk, x, r)$ **else** $\mathsf{Sim}(\mathsf{td}, pk, f(t, x))$
$B \leftarrow \mathcal{A}(Z)$
return B

The $\mathsf{EscHide}(\mathsf{PPB}, \mathcal{A}, \lambda)$ *advantage of* $\mathcal{A}$ *is defined as:*

$$\mathsf{Adv}^{\mathsf{EscHide}}_{\mathsf{PPB},\mathcal{A}}(\lambda) = \left|\Pr\left[\mathsf{EscHide}_0(\mathsf{PPB}, \mathcal{A}, \lambda) \Rightarrow 1\right] - \Pr\left[\mathsf{EscHide}_1(\mathsf{PPB}, \mathcal{A}, \lambda) \Rightarrow 1\right]\right|$$

A PPB satisfies escrow hiding if for all PPT adversaries $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{EscHide}}_{\mathsf{PPB},\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

3 An Efficient PPB For Threshold Comparison

We present here our main construction of an efficient PPB for threshold comparison. Let PKE_R and PKE_M be lossy public-key encryption schemes, and OTP, OTP' be one-time pad schemes, where $\mathsf{OTP}.\epsilon$ is some fixed key. Define the following two relations:

$$\mathcal{R}_{KG} = \left\{ ((pk', C), (t, sk', r_k, r_c)) \; : \; \begin{array}{c} (sk', pk') = \mathsf{KGen}'(\lambda, \Lambda_k^\mathsf{R}, \Lambda_k^\mathsf{M}, b, n, t; r_k) \\ \wedge\, C = \mathsf{Commit}(\Lambda_C, t; r_c) \end{array} \right\}$$
$$\mathcal{R}_E = \left\{ ((Z', C, pk), (v, m, r_e, r_c)) \; : \; \begin{array}{c} Z' = \mathsf{Escrow}'(\Lambda_k^\mathsf{R}, \Lambda_k^\mathsf{M}, n, pk, v, m; r_e) \\ \wedge\, C = \mathsf{Commit}(\Lambda_C, (v, m); r_c) \end{array} \right\}$$

Let $\mathcal{L}_{KG}$ and $\mathcal{L}_E$ be the corresponding languages to these relations, and Π_{KG}, Π_E be corresponding NIZKs for these languages. Our f_{thr}-PPB construction can then be seen in Figure 1 and Figure 2.

We can now state our main theorem.

Theorem 1. *If* OTP' *is a secure OTP scheme,* OTP *is an abelian OTP scheme with message space* $\mathcal{O}$*,* PKE_R *is a weakly robust, lossy, weak-lossy-ambiguous public-key encryption scheme,* PKE_M *is a lossy,* $\mathcal{U}_\mathcal{O}$*-strong-lossy-ambiguous public-key encryption scheme,* $(\mathsf{PKE}_\mathsf{R}, \mathsf{PKE}_\mathsf{M}, \mathsf{OTP})$ *satisfy lossy correlation resistance,* $\mathsf{C} = (\mathsf{CSetup}, \mathsf{Commit})$ *is a secure commitment scheme,* Π_{KG} *is zero-knowledge and strong simulation extractable, and* Π_E *is zero-knowledge and satisfies knowledge soundness, then* (ℓ, b)*-TCPPB is a secure* f_{thr}*-PPB scheme.*

Due to space constraints, proof of Theorem 1 can be found in the full version of the paper.

$\mathsf{Setup}(1^\lambda, \Lambda_C)$

$(\mathsf{crs}_{KG}, \mathsf{td}_{KG}) \leftarrow \mathsf{S}_{KG}(1^\lambda)$
$(\mathsf{crs}_E, \mathsf{td}_E) \leftarrow \mathsf{S}_E(1^\lambda)$
$\Lambda_k^{\mathsf{R}} \leftarrow \mathsf{PKE_R.Setup}(1^\lambda)$
$\Lambda_k^{\mathsf{M}} \leftarrow \mathsf{PKE_M.Setup}(1^\lambda)$
$n = \lceil \log_b(\ell) \rceil$
$\Lambda = (\lambda, \Lambda_C, \Lambda_k^{\mathsf{R}}, \Lambda_k^{\mathsf{M}}, \mathsf{crs}_{KG}, \mathsf{crs}_E, n)$
return $(\Lambda, (\mathsf{td}_{KG}, \mathsf{td}_E))$

$\mathsf{KGen}'\left(\lambda, \Lambda_k^{\mathsf{R}}, \Lambda_k^{\mathsf{M}}, n, t\right)$

for $i \in [n]$
  for $j \in [0, b]$
    if $j \leq t[i]$
      $rpk_i^j \leftarrow \mathsf{PKE_R.KGen_{loss}}(\Lambda_k^{\mathsf{R}})$
      $mpk_i^j \leftarrow \mathsf{PKE_M.KGen_{loss}}(\Lambda_k^{\mathsf{M}})$
      $\overline{sk}_i^j \leftarrow \perp$
    else
      $rsk_i^j, rpk_i^j \leftarrow \mathsf{PKE_R.KGen}(\Lambda_k^{\mathsf{R}})$
      $msk_i^j, mpk_i^j \leftarrow \mathsf{PKE_M.KGen}(\Lambda_k^{\mathsf{M}})$
      $\overline{sk}_i^j \leftarrow (rsk_i^j, msk_i^j)$
$sk' = \{\overline{sk}_i^j\}_{i \in [n], j \in [0, b-1]}$
$pk' = \{\overline{pk}_i^j\}_{i \in [n], j \in [0, b-1]}$
return (sk', pk')

$\mathsf{Escrow}(\Lambda, pk, (v, m), r_E)$

parse $(\lambda, \Lambda_C, \Lambda_k^{\mathsf{R}}, \Lambda_k^{\mathsf{M}}, \mathsf{crs}_{KG}, \mathsf{crs}_E, n) = \Lambda$
parse $(pk', C_{KG}, \pi_{KG}) = pk$
if $\mathsf{VerPK}(\Lambda, pk, C_{KG}) = 0$ **then return** $\perp$
$Z' \xleftarrow{r_e} \mathsf{Escrow}'(\Lambda_k^{\mathsf{R}}, \Lambda_k^{\mathsf{M}}, n, pk', v, m)$
$C_E \leftarrow \mathsf{Commit}(\Lambda_C, (v, m); r_E)$
$\pi_E \leftarrow \mathsf{P}_E(\mathsf{crs}_E, (Z', C_E, pk'), (v, m, r_e, r_E))$
return (Z', π_E)

$\mathsf{KGen}\,(\Lambda, t, r_{KG})$

parse $(\lambda, \Lambda_C, \Lambda_k^{\mathsf{R}}, \Lambda_k^{\mathsf{M}}, \mathsf{crs}_{KG}, \mathsf{crs}_E, n) = \Lambda$
$(sk', pk') \xleftarrow{r_k} \mathsf{KGen}'(\lambda, \Lambda_k^{\mathsf{R}}, \Lambda_k^{\mathsf{M}}, n, t)$
$C_{KG} \leftarrow \mathsf{Commit}(\Lambda_C, t; r_{KG})$
$\pi_{KG} \leftarrow \mathsf{P}_{KG}(\mathsf{crs}_{KG}, (pk', C_{KG}), (t, sk', r_k, r_{KG}))$
$pk \leftarrow (pk', C_{KG}, \pi_{KG})$
$sk \leftarrow (sk', pk)$
return (sk, pk)

$\mathsf{Escrow}'(\Lambda_k^{\mathsf{R}}, \Lambda_k^{\mathsf{M}}, n, pk, v, m)$

$\alpha^* \leftarrow \mathsf{OTP}'.\mathsf{KGen}()$
$\beta^* \leftarrow \mathsf{OTP}'.\mathsf{Enc}(\alpha^*, m)$
$\alpha_1, \ldots, \alpha_{n-1} \leftarrow \mathsf{OTP.KGen}()$
$\alpha_0 \leftarrow \mathsf{OTP}.\varepsilon$
for $i \in [n]$
  $ct_i^{\mathsf{R}} \leftarrow \mathsf{PKE_R.Enc}(\Lambda_k^{\mathsf{R}}, rpk_i^{v[i]}, \alpha^*)$
  $\beta_i^{\mathsf{R}} \leftarrow \mathsf{OTP.Enc}(\alpha_{i-1}, ct_i^{\mathsf{R}})$
  if $i < n$
    $\beta_i^{\mathsf{M}} \leftarrow \mathsf{OTP.Enc}(\alpha_{i-1}, \alpha_i)$
    $ct_i^{\mathsf{M}} \leftarrow \mathsf{PKE_M.Enc}(\Lambda_k^{\mathsf{M}}, mpk_i^{v[i]+1}, \beta_i^{\mathsf{M}})$
  else
    $ct_i^{\mathsf{M}} \leftarrow \perp$
  $\overline{ct}_i := (\beta_i^{\mathsf{R}}, ct_i^{\mathsf{M}})$
return $(\{\overline{ct}_i\}_{i \in [n]}, \beta^*)$

Fig. 1: Our f_{thr}-PPB construction (ℓ, b)-TCPBB. (ℓ, b) are assumed to be implicit inputs to each algorithm.

$\mathsf{VerPK}(\Lambda, pk, C_{KG})$

$\quad$ **parse** $(\lambda, \Lambda_C, \mathsf{crs}_{KG}, \mathsf{crs}_E, \Lambda_k, \ell, b, n) = \Lambda$
$\quad$ **parse** $(pk', C'_{KG}, \pi_{KG}) = pk$
$\quad$ **return** $\mathsf{V}_{KG}(\mathsf{crs}_{KG}, \pi_{KG}, (pk', C_{KG}))$
$\quad\quad \wedge (C'_{KG} = C_{KG})$

$\mathsf{VerEscrow}(\Lambda, pk, C_E, Z = (Z', \pi_E))$

$\quad$ **parse** $(\lambda, \Lambda_C, \mathsf{crs}_{KG}, \mathsf{crs}_E, \Lambda_k, \ell, b, n) = \Lambda$
$\quad$ **parse** $(pk', C'_{KG}, \pi_{KG}) = pk$
$\quad$ **return** $\mathsf{VerPK}(\Lambda, pk, C_{KG})$
$\quad\quad \wedge\ \mathsf{V}_E((Z', C_E, pk'), \pi_E)$

$\mathsf{Dec}(\Lambda, sk, C_E, Z = (Z', \pi_E))$

$\quad$ **parse** $(\lambda, \Lambda_C, \Lambda_k^{\mathsf{R}}, \Lambda_k^{\mathsf{M}}, \mathsf{crs}_{KG}, \mathsf{crs}_E, n) = \Lambda$
$\quad$ **parse** $(\{(rsk_i^j, msk_i^j)\}_{i \in [n], j \in [0,b]}, pk)$
$\quad\quad = sk$
$\quad$ **if** $\mathsf{VerEscrow}(\Lambda, pk, C_E, Z) = 0$ **return** $\perp$
$\quad$ **parse** $(\{\overline{ct}_i\}_{i \in [n]}, \beta^*) = Z'$
$\quad$ $\alpha'_0 := \mathsf{OTP}.\varepsilon$
$\quad$ **for** $i \in [n]$
$\quad\quad$ $(\beta_i^{\mathsf{R}}, ct_i^{\mathsf{M}}) \leftarrow \overline{ct}_i$
$\quad\quad$ $ct_i^{\mathsf{R}} \leftarrow \mathsf{OTP}.\mathsf{Dec}(\alpha'_{i-1}, \beta_i^{\mathsf{R}})$
$\quad\quad$ **for** $j \in [t[i] + 1, b]$
$\quad\quad\quad$ $\alpha^{*\prime} = \mathsf{PKE}_{\mathsf{R}}.\mathsf{Dec}(\Lambda_k^{\mathsf{R}}, rsk_i^j, ct_i^{\mathsf{R}})$
$\quad\quad\quad$ **if** $\alpha^{*\prime} \neq \perp$ **return** $\mathsf{OTP}.\mathsf{Dec}(\alpha^{*\prime}, \beta^*)$
$\quad\quad$ **if** $i < n$
$\quad\quad\quad$ $\beta_i^{\mathsf{M}} \leftarrow \mathsf{PKE}_{\mathsf{M}}.\mathsf{Dec}(\Lambda_k^{\mathsf{M}}, msk_i^{t[i]+1}, ct_i^{\mathsf{M}})$
$\quad\quad\quad$ $\alpha'_i \leftarrow \mathsf{OTP}.\mathsf{Dec}(\alpha'_{i-1}, \beta_i^{\mathsf{M}})$
$\quad$ **return** $\perp$

Fig. 2: Our f_{thr}-PPB construction continued.

3.1 Concrete Instantiation

We describe here a concrete instantiation of our PPB scheme using a lossy variant of the ElGamal encryption scheme [9]. Our lossy variant uses an algorithm $\mathcal{H}$ that takes as input a security parameter 1^λ and outputs a group description with *two* generators $G = (\mathbb{G}, p, g, h)$, where the discrete log of h with respect to g is unknown. The scheme generates lossy keys by running the traditional ElGamal key generation algorithm with the alternate generator h. As the first element of a ciphertext remains g^r, computational lossiness follows from DDH. We give a full description of this scheme and prove prove that it satisfies collision-freeness, computational lossiness, and $\mathcal{U}_\mathcal{O}$-strong-lossy-ambiguity in the full version of the paper.

Our scheme additionally requires a PKE with weak-robustness and weak-lossy-ambiguity. We therefore give a generic transformation from a collision-free PKE to a weakly robust PKE that preserves lossiness and weak-lossy-ambiguity. The transform uses two public keys, and then has ciphertexts encrypt a public flag value ϕ under the second key. The transform clearly preserves lossiness, and we prove that it provides weak-robustness while preserving weak-lossy-ambiguity in the full version of the paper.

The final piece we need to instantiate our construction is suitable one-time-pad schemes. For OTP', we will use the natural scheme where a key α consists of a randomly sampled group element and messages are blinded via the group operation. For completeness, we provide a description of this scheme in the full version of the paper. The other OTP scheme is applied to two objects in our construction: $\mathsf{PKE_R}$ ciphertexts and OTP keys. In Section 3.2 we will show an optimization that allows $\mathsf{PKE_R}$ ciphertexts to be represented with only two group elements. Therefore, we will have OTP consist of two copies of OTP' run in parallel.

We can then instantiate the scheme in Figure 1 as follows, where the base scheme is the lossy ElGamal variant described above:

- $\mathsf{PKE_R}$ is the scheme that results from applying the weak-robust transform to the base scheme.
- $\mathsf{PKE_M}$ is two copies of the base scheme run in parallel.
- OTP' is the scheme described above and shown in **??**, and OTP is two copies of OTP' run in parallel.

3.2 Optimizations

We describe here optimizations that reduce the public-key and escrow size of the concrete PPB scheme described in Section 3.1 while preserving security.

Randomness Reuse. As written, a $\mathsf{PKE_R}$ public-key would consist of two sub-keys $pk_0 = g^{x_0}, pk_1 = g^{x_1}$, and a ciphertext would consist of two ElGamal encryptions of the form $((g^y, g^{x_0 y} \cdot m), (g^z, g^{x_1 z} \cdot \phi))$. We can compress this ciphertext via randomness re-use. ElGamal encryption allows an encryptor to re-use the same

g^r value across encryptions under different public-keys. This was first shown to be secure in [17], and later tested securely under a more stringent security definition in [4]. We can leverage this to use the same g^{r_R} value across *all* reveal ciphertexts, as each is under a different key. $\mathsf{PKE_R}$ ciphertexts now take the form: $(g^{x_0 r_\mathsf{R}} \cdot m, g^{x_1 r_\mathsf{R}} \cdot \phi)$, with g^{r_R} included separately in the escrow value.

Shared Public Keys. As our instantiated OTP scheme has keys α that consist of two group elements (α_1, α_2), match ciphertexts ct^M must consist of *two* ElGamal encryptions. For these, we can use the same (pk_0, pk_1) present in the $\mathsf{PKE_R}$ public key. We can additionally use the same randomness-reuse trick as before, using the same random value r_M for all match ciphertexts. A match ciphertext ct^M encrypting an OTP value $\alpha = (\alpha_0, \alpha_1)$ then has the form: $(g^{x_0 r_\mathsf{M}} \cdot \alpha_0, g^{x_1 r_\mathsf{M}} \cdot \alpha_1)$, with r_M included separately in the escrow value. With this optimization the scheme's public-key only needs to include two ElGamal public-keys in each cell, rather than four as before.

4 Implementation

As the goal of our construction is an efficient Escrow algorithm, our implementation and experiments focus on this portion of the schemes. Code for our implementations can be found at `https://github.com/anonfc2026/efficient-ppb`.

Our implementation for (ℓ, b)-TCPPB uses the concrete algorithm instantiations described in Section 3.1 and the optimizations described in Section 3.2. For proof generation we implement Escrow' in Circom [5] version 2.2.2, using the Baby JubJub elliptic curve [32] as the underlying group for the encryption schemes. Proofs were generated using snarkjs [27].

The baseline scheme we compare against is the NISC-based PPB described in [13]. We instantiate the NISC with the well-known construction based on garbled circuits [33]. In this scheme, the auditor's public key consists of n oblivious transfer (OT) receiver messages, where n is bit length of the integers to be compared. An escrow is then a garbled n-bit comparison circuit, the labels corresponding to the user's input, and the OT sender messages encoding the remaining labels. We use the Free-XOR optimized comparator circuit from [14]. For completeness, we give a description of OT, garbled circuits, and this scheme in the full version of the paper.

A notable downside of this construction is that proving over the garbling algorithm requires non-black-box use of random oracles due to optimizations like Free-XOR [15] and Half-Gates [34]. Despite this caveat we have implemented this approach to provide the closest possible alternative to our main PPB scheme, giving it the best opportunity to serve as a viable comparison. We implement standard garbling with the Free-XOR and Half-Gates optimizations, and use Poseidon [10] as our hash function. Wire labels are stored as 254-bit arrays and converted to field elements before hashes are performed, with the resulting digest then converted back to a 254-bit array.

5 Experiments

Experiments were run on a desktop computer with an AMD Ryzen 5 5600X CPU and 32GB of RAM. We test our schemes on implementations for comparing integers up 2^{32} and 2^{64}. For (ℓ, b)-TCPPB we use $b = 41$, which results in $n = 6$ in the first case and $n = 12$ in the second.[6] The results of our experiments can be seen in Table 2. Our (ℓ, b)-TCPPB gives a $\geq 8\times$ improvement in escrow size, along with a $\geq 2\times$ improvement in prover time as compared to the baseline scheme.

Table 2: Escrow sizes and prover times of the baseline NISC-based PPB and the TCPPB presented in this work. The TCPPB uses base $b = 41$.

Scheme	ℓ	**Escrow Size (bytes)**	**Prover Time (seconds)**
Garbled Circuit	2^{32}	13,472	11.95
	2^{64}	26,784	30.06
TCPPB	2^{32}	1,600	4.42
	2^{64}	3,136	7.86

References

[1] Michel Abdalla, Mihir Bellare, and Gregory Neven. “Robust Encryption”. In: *TCC 2010*. Ed. by Daniele Micciancio. Vol. 5978. LNCS. Springer, Berlin, Heidelberg, Feb. 2010, pp. 480–497. DOI: 10.1007/978-3-642-11799-2_28.

[2] Prabhanjan Ananth and Abhishek Jain. “Pre-constrained encryption”. In: *13th Innovations in Theoretical Computer Science Conference*. 2022.

[3] Gabrielle Beck et al. “Fuzzy Message Detection”. In: *ACM CCS 2021*. Ed. by Giovanni Vigna and Elaine Shi. ACM Press, Nov. 2021, pp. 1507–1528. DOI: 10.1145/3460120.3484545.

[4] Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. “Randomness Re-use in Multi-recipient Encryption Schemeas”. In: *PKC 2003*. Ed. by Yvo Desmedt. Vol. 2567. LNCS. Springer, Berlin, Heidelberg, Jan. 2003, pp. 85–99. DOI: 10.1007/3-540-36288-6_7.

[5] Marta Bellés-Muñoz et al. “Circom: A circuit description language for building zero-knowledge applications”. In: *IEEE Transactions on Dependable and Secure Computing* 20.6 (2022), pp. 4733–4751.

[6] Ilaria Chillotti et al. “TFHE: fast fully homomorphic encryption over the torus”. In: *Journal of Cryptology* 33.1 (2020), pp. 34–91.

[6] We set $b = 41$ as it represents a reasonable tipping point for the escrow size: $\lceil \log_{40}(2^{32}) \rceil = 7$, $\lceil \log_{41}(2^{32}) \rceil = 6$.

[7] CoinDesk. *Dubai Prohibits Privacy Coins Like Monero Under New Crypto Rules*. `https://www.coindesk.com/policy/2023/02/08/dubai-prohibits-privacy-coins-under-new-crypto-rules/`. 2023.

[8] Bernardo David et al. "Updatable Privacy-Preserving Blueprints". In: *ASIACRYPT 2024, Part I*. Ed. by Kai-Min Chung and Yu Sasaki. Vol. 15484. LNCS. Springer, Singapore, Dec. 2024, pp. 105–139. DOI: `10.1007/978-981-96-0875-1_4`.

[9] Taher ElGamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *IEEE transactions on information theory* 31.4 (1985), pp. 469–472.

[10] Lorenzo Grassi et al. "Poseidon: A new hash function for {Zero-Knowledge} proof systems". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 519–535.

[11] Scott Griffy et al. *Privacy-Preserving Blueprints via Succinctly Verifiable Computation over Additively-Homomorphically Encrypted Data*. Cryptology ePrint Archive, Report 2024/675. 2024. URL: `https://eprint.iacr.org/2024/675`.

[12] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *EUROCRYPT 2016, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. LNCS. Springer, Berlin, Heidelberg, May 2016, pp. 305–326. DOI: `10.1007/978-3-662-49896-5_11`.

[13] Markulf Kohlweiss, Anna Lysyanskaya, and An Nguyen. "Privacy-Preserving Blueprints". In: *EUROCRYPT 2023, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. LNCS. Springer, Cham, Apr. 2023, pp. 594–625. DOI: `10.1007/978-3-031-30617-4_20`.

[14] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. "Improved garbled circuit building blocks and applications to auctions and computing minima". In: *International Conference on Cryptology and Network Security*. Springer. 2009, pp. 1–20.

[15] Vladimir Kolesnikov and Thomas Schneider. "Improved garbled circuit: Free XOR gates and applications". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2008, pp. 486–498.

[16] Ahmed Kosba et al. "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts". In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 839–858.

[17] Kaoru Kurosawa. "Multi-recipient Public-Key Encryption with Shortened Ciphertext". In: *PKC 2002*. Ed. by David Naccache and Pascal Paillier. Vol. 2274. LNCS. Springer, Berlin, Heidelberg, Feb. 2002, pp. 48–63. DOI: `10.1007/3-540-45664-3_4`.

[18] Sarah Meiklejohn et al. "A fistful of bitcoins: characterizing payments among men with no names". In: *Proceedings of the 2013 conference on Internet measurement conference*. 2013, pp. 127–140.

[19] P. E. Meltzer. "Keeping Drug Money from Reaching the Wash Cycle: A Guide to the Bank Secrecy Act". In: *Banking Law Journal* 108.3 (May 1991). NCJ Number: 132406, pp. 230–255.

[20] Payman Mohassel. "A Closer Look at Anonymity and Robustness in Encryption Schemes". In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Berlin, Heidelberg, Dec. 2010, pp. 501–518. DOI: 10.1007/978-3-642-17373-8_29.

[21] Shen Noether, Adam Mackenzie, et al. "Ring confidential transactions". In: *Ledger* 1 (2016), pp. 1–18.

[22] Jean-Baptiste Orfila, Arthur Meyre, and Agnes Leroy. *TFHE-rs v0.7: Ciphertext Compression, Multi-GPU Support and More*. Zama blog. urlhttps://www.zama.ai/post/tfhe-rs-v0-7-ciphertext-compression-multi-gpu-support-and-more. July 2024.

[23] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. "A framework for efficient and composable oblivious transfer". In: *Annual international cryptology conference*. Springer. 2008, pp. 554–571.

[24] Alexey Pertsev, Roman Semenov, and Roman Storm. "Tornado cash privacy solution version 1.4". In: *Tornado cash privacy solution version* 1.6 (2019).

[25] Eli Ben Sasson et al. "Zerocash: Decentralized anonymous payments from bitcoin". In: *2014 IEEE symposium on security and privacy*. IEEE. 2014, pp. 459–474.

[26] Jack Schickler. "Privacy-Enhancing Crypto Coins Could Be Banned Under Leaked EU Plans". In: *CoinDesk* (2022). URL: https://www.coindesk.com/policy/2022/11/15/privacy-enhancing-crypto-coins-could-be-banned-under-leaked-eu-plans/.

[27] *snarkjs*. Version 0.7.5. 2019. URL: https://github.com/iden3/snarkjs.

[28] U.S. Department of the Treasury. *U.S. Treasury Sanctions Notorious Virtual Currency Mixer Tornado Cash*. https://home.treasury.gov/news/press-releases/jy0916. Press release. 2022.

[29] U.S. Department of the Treasury. *U.S. Treasury Sanctions Notorious Virtual Currency Mixer Tornado Cash*. Press Release. Aug. 2022. URL: https://home.treasury.gov/news/press-releases/jy0916.

[30] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. "Verifiable fully homomorphic encryption". In: *arXiv preprint arXiv:2301.07041* (2023).

[31] Robert Viglione. "Japan's Ban Is a Wake-Up Call to Defend Privacy Coins". In: *CoinDesk* (2018). URL: https://www.coindesk.com/markets/2018/05/29/japans-ban-is-a-wake-up-call-to-defend-privacy-coins/.

[32] Barry WhiteHat, Jordi Baylina, and Marta Bellés. "Baby Jubjub elliptic curve". In: *Ethereum Improvement Proposal, EIP-2494* 29 (2020), p. 27.

[33] Andrew C Yao. "Protocols for secure computations". In: *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE. 1982, pp. 160–164.

[34] Samee Zahur, Mike Rosulek, and David Evans. "Two halves make a whole: Reducing data transfer in garbled circuits using half gates". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2015, pp. 220–250.