Truncator: Time-space Tradeoff of Cryptographic Primitives

Foteini Baldimtsi[1,2], Konstantinos Chalkias[2], Panagiotis Chatzigiannis[3], and Mahimna Kelkar[4]

[1] George Mason University `foteini@gmu.edu`
[2] Mysten Labs `kostas@mystenlabs.com`
[3] Visa Research** `pchatzig@visa.com`
[4] Cornell University `mahimna@cs.cornell.edu`

Abstract. We're presenting mining-based techniques to reduce the size of various cryptographic outputs without loss of security. Our approach can be generalized for multiple primitives, such as key generation, signing, hashing and encryption schemes, by introducing a brute-forcing step to provers/senders aiming at compressing submitted cryptographic material. As a result, in systems that we can tolerate sender's work to be more demanding and time-consuming, we manage to optimize on verification, payload size and storage cost, especially when:

- receivers have limited resources (i.e. mobile, IoT);
- storage or data-size is financially expensive (i.e. blockchains, cloud storage and ingress cost);
- multiple recipients perform verification/decryption/lookup (i.e. blockchains, TLS certs, IPFS lookups).

Interestingly, mining can result in record-size cryptographic outputs, and we show that 5%-12% shorter hash digests and signatures are practically feasible even with commodity hardware. Obviously, the first thing that comes to mind is compressing addresses and transaction signatures in order to pay less gas fees in blockchain applications, but in fact even traditional TLS certificates and public keys, which are computed once and reused in every new connection, can be slightly compressed with this "mining" trick without compromising security. The effects of "compressing once - then reuse" at mass scale can be economically profitable in the long run for both the Web2 and Web3 ecosystems.

Our paradigm relies on a brute-force search operation in order to craft the primitive's output such that it fits into fewer bytes, while the "missing" fixed bytes are implied by the system parameters and omitted from the actual communication. While such compression requires computational effort depending on the level of compression, this cost is only paid at the source (typically in blockchains consisting of a single party) which is rewarded by lowered transaction fees, and the benefits of the compression are enjoyed by the whole ecosystem. As a starting point, we show how our paradigm applies to some basic primitives (commonly used in blockchain applications), and show how security is preserved using a bit security framework. Surprisingly, we also identified cases where wise mining strategies require proportionally less effort than naive brute-forcing, an example is WOTS [12] (and inherently SPHINCS [10]) post-quantum signatures where the target goal is to remove or compress the Winternitz checksum part. Moreover, we evaluate our approach for several primitives based on different levels of compression which concretely demonstrates the benefits (both in terms of financial cost and storage) if adopted by the community.

Finally, as this work is inspired by the recent unfortunate buggy "gas golfing" software in Ethereum, where weakly implemented functions incorrectly generated addresses (hashes) with "prefixed zeroes for gas optimization", resulting in millions of losses [1,18,6], we expect our *Truncator* approach to be naturally applied in the blockchain space as a secure solution to more succinct transactions, addresses and states.

1 Introduction

In blockchain applications like Bitcoin [20], a distributed common ledger is maintained among all participants. As the size of the ledger monotonically increases, most blockchains have large storage requirements for nodes, which can be several hundreds of gigabytes, even after applying techniques to prune or compact the needed storage. In addition, there is typically an upper bound of storage

** Panagiotis Chatzigiannis did part of this work during his PhD studies at George Mason University.

space per block (e.g. in Bitcoin it is 1MB)[5], while concretely for Bitcoin at the time of writing, 250 bytes roughly cost $1.5 of transaction fees to include in next block with high probability. Therefore, blockchain space is a scarce resource, and typically there are mechanisms in place to disincentivize posting large amounts of data to the public ledger (e.g. in cryptocurrencies, transaction fees are proportional to the size of that transaction in bytes).

In this work, we present several methods to trade off storage for computation in several cryptographic primitives used in both Web2 (i.e. succinct TLS keys and certificates) and blockchain applications. Our approach is based on the principle that a few extra operations on behalf of the transaction's sender, which could require crafting a valid transaction in a few seconds or minutes instead of milliseconds, would be beneficial for the sender (by lowering transaction fees) as well as all other blockchain participants (by fitting more transactions per block or by reducing the overall blockchain size by a constant factor). Our underlying paradigm for these operations is a brute-force search in the cryptographic primitive's inputs/randomness, in order to craft each primitive's output in a specific way that satisfies the modified system's public parameters, e.g. requiring some specific bits of the output to be constant. This enables us to omit these bits from the output entirely, as these implied constant bits can be "glued back" to the output by the receiver, effectively allocating fewer bits per such output for communication and storage. For each primitive, we argue that security is preserved compared to the standard primitive.

Finally, while in this paper we focus on the basic cryptographic primitives commonly used in the blockchain space (where storage costs are of particularly importance), our techniques can be further applied to the whole spectrum of cryptography (e.g. zero-knowledge proofs, lattices, multi-party computation etc.) with the level of potential benefits depending on the specific application where these primitives are deployed.

1.1 Overview of our approach

As discussed above, our approach will be an iterative search of the primitive's input such that the conditions we require for the primitive's output are satisfied. As a first example, in the key generation algorithm for *dlog*-based keys, we perform an iterative search a secret key sk such that its derived public key $\mathsf{pk} = g^{\mathsf{sk}}$ has a pre-determined ℓ-bit prefix. On other probabilistic primitives, e.g. in a public key encryption scheme, we can simply brute-force the scheme's randomness to achieve the desired truncation. However if we need to truncate a deterministic primitive (e.g. a hash function), a nonce (or salt) must be used. Another possible technique is to introduce some randomness within the primitive's payload without altering its semantics, e.g. slightly altering some pixels in images to nearby colors, or replacing spaces with non-printable characters in text files. The latter approach is straightforward and easy to implement, without directly modifying the cryptographic primitive, but is only applicable within certain application scenarios.

Based on the above, we distinguish between the 2 main ways of a brute-force search on a primitive's input: brute-forcing the internal randomness of a primitive (if any) or brute-forcing the primitive's payload (e.g. the message of a signature). Brute-forcing the payload can be implemented in 3 different ways:

- Use a nonce, and send it along with the payload. This method is preferable if the application already includes such a nonce.
- Use a nonce, and have the receiver perform a brute-force operation as well to recover it. A similar method was recently proposed by Pornin for signatures [23], but this is not suitable for our applications as there we try to optimize on the verifier's side.

[5] In Ethereum there is no upper bound in block size, but each block has a maximum total gas, which has a similar effect.

– Brute-force the payload directly without changing it semantics, e.g. slightly altering pixels in images, use non-printable chars instead of spaces in documents etc.

Randomness search. In the case where we perform a brute-force search on a primitive's randomness, it is particularly important on how this search is algorithmically performed. Simply incrementing the randomness might lead to potential attacks in some applications, (e.g. in RSA, two random values having a difference of 1 might result in the same key pair from the primality checks) and the safest way is to generate fresh randomness for each iteration [7,8]. While this is a bit costlier in terms of computation (because it needs to invoke `/dev/urandom` each time), a potential cheaper alternative would be to increment by a large constant instead (although this needs to be carefully considered for each primitive). For the case of hash functions, randomness generation could be performed similarly as in Randomized Hashing for Digital Signatures [16]. Finally, we highlight a recent attack on an Ethereum vanity address generator [4], where the randomness for brute-forcing the prefix on addresses was only 32 bits, making a reverse brute-force search to recover the corresponding private keys feasible, which in turn led to loss of funds [6].

The role of bit security. In our work, we will use a recent bit security framework [25] to analyze the security of our proposed scheme modifications. Bit-security, is commonly used to describe the level of security offered by a concrete instantiation of a cryptographic primitive P and offers a middle ground approach between the common asymptotic proof approach and the concrete security approach. Informally, we say that P has κ-bit security if it takes an adversary 2^κ operations to break it, or alternatively, an efficient adversary breaks the scheme with at most $\epsilon < 2^{-\kappa}$ probability. This implies that for any attack with computational cost T and success probability ϵ, it must hold that $T/\epsilon > 2^\kappa$. Intuitively, bit-security captures that P is as secure as an idealized perfect cryptographic primitive with an κ-bit key.

Our results. We show how to apply our truncation paradigm on some common cryptographic primitives, such as hash digests, ECC public keys and signature outputs, resulting in about 7% compression (2 bytes less) in less than a second for ed25519 signatures, and less than 10 milliseconds for compressed Blake3 digests, using our optimized Rust *Truncator* implementation [13]. Using the framework by Watanabe and Yasunaga [25], we show that bit security of the original primitive is preserved after our modifications, and we evaluate the computational overhead compared to the communication/storage savings. In addition, we consider primitives that involve an auxiliary output such as Winternitz one-time signatures, and we show how our paradigm has the potential to be even more efficient when applied in these cases. Note that these "truncated" versions of the primitives we considered only serve as a starting point, as our paradigm can be applied to the whole space of cryptography.

1.2 Related works

Perhaps the first technique in the blockchain space that resembles our paradigm is Bitcoin vanity address generator [5], which attempts to create a new valid Bitcoin public address (i.e. a double-hashed ECDSA public key) given a user-specified address prefix. Later, this approach was leveraged to create slightly shorter signatures in Bitcoin [2,3]. A more recent work by Pornin [23] presented techniques to reduce the size of EdDSA and ECDSA signatures, however these techniques required computational work on behalf of the verifier. Ethereum developers also proposed the use of addresses with a prefix of many zeroes in order to reduce gas fees (called "gas golfing") [1]. Also a recent work by Fleischhacker et al. [17] presented algorithms on compressing sparsely-encrypted vectors. Finally, a recent work by Blocki and Lee [11] which showed how to compress Schnorr signatures. However this type of compression might affect security [14]. Nevertheless, our approach is orthogonal and can be applied on top of such compression.

2 Preliminaries

Notation. We denote a probabilistic polynomial-time (PPT) algorithm B with input a and output b as $b \xleftarrow{\$} B(a)$. We denote the security parameter by λ, the bit security by κ and the truncation parameter (i.e. the number of bits truncated) by ℓ.

2.1 Computational Hardness Assumptions

Definition 1 (Discrete-logarithm problem). *The discrete-logarithm problem for a cyclic group $\mathbb{G}$ of order $q = |\lambda|$ is hard if $\forall$ ppt algorithms $\mathcal{A}$, $\exists$ negligible function negl s.t.:*

$$\Pr\left[\begin{array}{l} g \text{ generator of } \mathbb{G}; \\ h \xleftarrow{\$} \mathbb{G}; \\ x \leftarrow \mathcal{A}(g, h); \\ \text{if } g^x = h \text{ output } 1, \text{else output } 0 \end{array}\right] = 1 \leq negl(\lambda)$$

2.2 Definitions of Cryptographic Primitives

Definition 2 (Hard Relation). *A relation R with a randomized PPT sampling algorithm* Gen *is a hard relation if:*

- *For any $(x, y) \xleftarrow{\$} \mathsf{Gen}()$ we have $(x, y) \in R$.*
- *$\exists$ a PPT algorithm that decides if $(x, y) \in R$.*
- *$\forall$ PPT algorithms $\mathcal{A}$,* $\Pr\Big[(x, y) \xleftarrow{\$} \mathsf{Gen}(); x^* \leftarrow \mathcal{A}(y); R(x^*, y) = 1\Big] \leq negl(\lambda)$

2.3 Bit security of cryptographic primitives

We use the bit-security framework defined in recent work by Watanabe and Yasunaga [25] and provide an brief overview here (the alternative framework from [19] can also be used instead and gives the same results).

Basic intuition. Abstractly, if a cryptographic primitive has κ-bit security, then the intuition is that any adversary would need at least 2^κ operations to break it where the computational cost comes from the security game played by the adversary and the challenger. To precisely quantify bit security, the framework models two adversaries: an *inner* adversary $\mathcal{A}$ which plays the "usual" security game against the challenger, and an *outer* adversary $\mathcal{B}$ which invokes $\mathcal{A}$ a total of $N_{\mathcal{A},\mathcal{B}}$ times in order to amplify its final winning probability $\epsilon_{\mathcal{A},\mathcal{B}}$.

In an κ-bit security game, the challenger chooses a secret $u \in \{0, 1\}^\kappa$ uniformly at random, and sends the challenge $X(u)$ to $\mathcal{A}$. The game is classified as a *search* game when $u >> 1$, and as a *decision* game when $u = 1$. For instance, in the IND-CPA security game $\mathcal{A}$'s goal is to distinguish between two encryptions (i.e. u is 0 or 1) while in the simple discrete-logarithm experiment the adversary's goal is to output the value of u for a challenge g^u. Based on this distinction, the framework's structure is somewhat different according to the security game type. In search games, each inner adversary is invoked with a fresh random secret u, and the probability that $\mathcal{B}$ wins is defined as the probability that *some* $\mathcal{A}$ wins (i.e., finds the appropriate search quantity). In contrast, for decision games, each inner adversary plays an independent game with consistent secret u across all invocations. $\mathcal{B}$ can use the outputs from all inner adversaries to produce its output u'; its probability of winning can now be defined as $\Pr[u = u']$.

Definition 3 (Bit Security [25]). *The bit security of an κ-bit game* G *is defined by:*

$$\begin{aligned}\mathrm{BS}^{\mu}_{\mathrm{G}} &= \min_{\mathcal{A},\mathcal{B}} \{\log_2(N_{\mathcal{A},\mathcal{B}} \cdot T_{\mathcal{A}}) : \epsilon_{\mathcal{A},\mathcal{B}} \geq 1-\mu\} \\ &= \min_{\mathcal{A}} \left\{ \log_2(T_{\mathcal{A}}) + \log_2 \left(\min_{\mathcal{B}} \{N_{\mathcal{A},\mathcal{B}} : \epsilon_{\mathcal{A},\mathcal{B}} \geq 1-\mu\} \right) \right\}\end{aligned}$$

where $N_{\mathcal{A},\mathcal{B}}$ is the number of instances of $\mathcal{A}$ invoked by $\mathcal{B}$, $T_{\mathcal{A}}$ is the computational complexity for playing the inner game by $\mathcal{A}$ and $\mu > 0$ some small constant for $\mathcal{B}$ success probability $1-\mu$.

Based on the above definition and Theorems 1 and 2 provided in [25], the framework for a primitive with a search-type game provides an approximate bit security of $\kappa = \log_2(T_{\mathcal{A}}/\epsilon_{\mathcal{A}})$ where $\epsilon_{\mathcal{A}}$ is $\mathcal{A}$'s success probability in the security game. For primitives with decision-type games (e.g. PRG, encryption, DDH) the framework approximates a lower bound for $\kappa \geq \log_2(T_{\mathcal{A}}/\delta)$ where δ is the advantage of $\mathcal{A}$ when playing a decision game.

3 Truncating cryptographic primitive outputs

Using the bit-security framework from Section 2.3, we now show how a number of different primitives can be compressed or truncated without affecting their concrete security.

3.1 Truncated Hash Functions

We first consider the truncation of simple hash functions. Let $H : \{0,1\}^* \to Y = \{0,1\}^\lambda$ be a (cryptographic) hash function. Suppose that we wish to communicate the λ-bit hash output $H(x)$ of an input x. To compress the amount of communication required, we now define a truncated hash function H'. For truncation parameter ℓ, we define H' as a function from $\{0,1\}^*$ to $\{0,1\}^{\lambda-\ell}$; this will intuitively denote the output of H truncated by ℓ bits. In particular, we start by fixing an ℓ-bit string $s = s_1, \ldots, s_\ell$. Let $Y_s \subseteq Y$ denote the subset of Y that contains exactly those outputs that begin with s. We consider the prefix for simplicity, but in general the truncated bits can be in any positions and do not have to be consecutive—this would also support e.g., truncation to a subgroup of the output group. Intuitively, our truncated function $H' = H'_s$ will now sample a nonce r such that $y = H(r \parallel x) \in Y_s$; this will be taken as the output of H'.

The upshot is that now the first ℓ bits do not need to be communicated as part of the hash output; this can be done since the string s, while part of the hash output, will be publicly known to the receiver and therefore assumed to be implicit without needing to be communicated.

Truncated Hashing

```
$H'_s(x)$:
For $r = 0, 1, 2, \ldots$:
    $y \leftarrow H(r \parallel x)$
    If $y \in Y_s$, then output $y$
```

Fig. 1. Truncated Hash Function

We now argue that the bit-security is preserved despite outputting fewer bits of the digest.

Theorem 1. *Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ a hash function with κ-bit security and computational cost T_H. Then the truncated hash function $H'_s : \{0,1\}^* \to \{0,1\}^\lambda$ where the first ℓ bits are fixed to string $s \in \{0,1\}^\ell$ is also κ-bit secure.*

Proof (Sketch). Applying the bit-security paradigm from Definition 3 to the hash function H, let $\mathcal{A}$ denote the inner adversary that minimizes $\log_2(T_\mathcal{A}/\epsilon_\mathcal{A})$ where $T_\mathcal{A}$ denotes the computational complexity of $\mathcal{A}$ and $\epsilon_\mathcal{A}$ denotes its success probability. Then $\kappa = \log_2(T_\mathcal{A}/\epsilon_\mathcal{A})$. We now need to show that the truncated hash function H'_s also provides κ bits of security. For this, first, notice that an evaluation of H'_s takes on expectation 2^ℓ times the cost to evaluate H. However, since the output is also truncated by ℓ, the success probability of the inner adversary will also increase by a factor of 2^ℓ. This means that the expected bit-security of any $\mathcal{A}'$ playing the game for H will be given by $\log_2((T_{\mathcal{A}'}/2^\ell)/(\epsilon_{\mathcal{A}'}/2^\ell) = \log_2(T_{\mathcal{A}'}/\epsilon_{\mathcal{A}'})$. Since this value is minimized for adversary $\mathcal{A}$, we obtain that the bit-security of H' will also be $\log_2(T_\mathcal{A}/\epsilon_\mathcal{A}) = \kappa$.

3.2 Truncated DL-based public keys

Let $\mathbb{G}$ be a cyclic group of prime order p and g be a generator of $\mathbb{G}$. The key generation algorithm, $\mathsf{KeyGen}(1^\lambda)$ for DL based keys works as follows: $\mathsf{sk} = x, \mathsf{pk} = g^x$ where $x \overset{\$}{\leftarrow} \mathbb{Z}_p$.

Let $y_1 y_2 \dots y_\lambda$ denote the binary representation of pk which forms the output space for public keys Y. For truncation parameter $\ell < \lambda$, we fix an ℓ-bit string $s = s_1, \dots, s_\ell$. Let $Y_s \subseteq Y$ denote the subset of Y that contains exactly those outputs that begin with s. (As with hash functions above, we consider truncation on the prefix for simplicity.) The new key generation algorithm $\mathsf{KeyGen}'(1^\lambda)$ now works as follows: $x \overset{\$}{\leftarrow} \mathbb{Z}_p$, compute $\mathsf{pk} = g^x$, if $\mathsf{pk} \notin Y_s$ repeat by picking a new x, else output $\mathsf{pk} \in Y_s$. We note that for the case of public keys, while sampling a "valid" private key requires this overhead of additional repeated "brute-force" style operations, the space savings are *permanent* for the key's lifetime, which makes this compression attractive particularly for blockchain applications.

Theorem 2. *Let $\mathsf{KeyGen}(1^\lambda)$ be a key generation algorithm for the DL hard relation with κ-bit security and computational cost $T_\mathcal{A}$. Then the truncated key generation algorithm $\mathsf{KeyGen}'(1^\lambda)$ where the first ℓ bits are fixed to string $s \in \{0,1\}^\ell$ is also κ-bit secure.*

Proof (Sketch). The key generation algorithm for DL based keys is a hard relation (as defined in Def. 2) under the DL problem. A hard relation is a search-type game which provides bit security of $\log_2(T_\mathcal{A}/\epsilon_\mathcal{A})$ where $\epsilon_\mathcal{A}$ is $\mathcal{A}$'s success probability in the security game and $T_\mathcal{A}$ is the computational cost of the adversary. We argue that our truncated algorithm maintains the same bit security as the one offered by the underlying group $\mathbb{G}$ [6].

Without loss of generality, assume that the truncation parameter $\ell = \lambda - 1$ and thus the size of the public key is a single bit. Then, our truncated key sampling method for public key space Y_s as defined above creates a secret key space X' where $|X'| = 2$. However, an adversary cannot efficiently compute X' as this would directly reduce to breaking the DL assumption with non-negligible probability.

We now need to show that the truncated key generation algorithm also provides κ bits of security. If x's are sampled uniformly at random, computing the truncated pk will take on expectation 2^ℓ time. However, since the possible secret key space is also truncated by ℓ, the success probability of the inner adversary will also increase by a factor of 2^ℓ. This means that the expected bit-security of any $\mathcal{A}'$ playing the game for KeyGen' will be given by $\log_2((T_{\mathcal{A}'}/2^\ell)/(\epsilon_{\mathcal{A}'}/2^\ell) = \log_2(T_{\mathcal{A}'}/\epsilon_{\mathcal{A}'})$.

[6] We note that for the case of `secp256k1` discrete log keys, if the group size is λ-bits, then the probability of success for the adversary is roughly $1/2^{\lambda/2}$ and thus the bit security is $\lambda/2$-bits, i.e. in `secp256k1` a 256-bit key roughly offers 128-bit security [21].

3.3 Truncated Schnorr Signatures

We now consider compression on DL-based signatures, and we show an application of our approach on Schnorr signatures as an example. In Figure 2 we present a version of our truncated Schnorr signature (we use red font to indicate the differences from standard Schnorr Signatures).

As before, let $\mathbb{G}$ be a cyclic group of prime order p and g be a generator of $\mathbb{G}$. Let $H()$ be a random oracle implemented with a hash function that outputs a uniformly random element $e \in \mathbb{Z}_p$ where p is a prime of size 2λ-bits (in order to achieve λ-bit security)[7]. We truncate the hash function in the same way as in Sec. 3.1 and we denote the truncation string by s' given as input to the signing algorithm. The resulting signature $\sigma = (s, e)$ will save ℓ bits (the truncation parameter).

Note that while our truncation approach could alternatively be applied to the value s, this is not efficient since e is computed first during Sign. Also, attempting to truncate signature values e and s simultaneously would exponentially increase the truncation time. Therefore in general, for primitives where the output consists of two or more elements, it is recommended to truncate the element that is computed first in the algorithm.

Also note that generally in truncated signature schemes the space savings can be "stacked" with the savings from truncated public keys as well.

```
$\mathsf{KeyGen}(1^\lambda)$
$x \xleftarrow{\$} \mathbb{Z}_p$
$\mathsf{sk} \leftarrow x$
$\mathsf{pk} \leftarrow g^x$
**return** $(\mathsf{pk}, \mathsf{sk})$
```

```
$\mathsf{Sign}(\mathsf{sk}, m, s')$
1. $r \xleftarrow{\$} \mathbb{Z}_p$
2. $I \leftarrow g^r$
3. $e \leftarrow H(I \parallel m)$
4. If $e \notin Y_s$ return to step 1.
5. $s \leftarrow r + \mathsf{sk} \cdot e \bmod p$
6. **return** $\sigma = (s, e)$
```

```
$\mathsf{Verify}(\sigma, m, \mathsf{pk})$
Parse $\sigma = (s, e)$
$I \leftarrow g^s \mathsf{pk}^{-e}$
If $(H(I \parallel m) = e)$ then
    **return** 1
else **return** 0
```

Fig. 2. The truncated Schnorr Signature Scheme

The security of Schnorr signatures (existential unforgeability) has been thoroughly analyzed in the literature [22,24]. We now argue that our truncated Schnorr signature scheme maintains the same bit security as the underlying non-truncated version. The theorem below is straightforward given Theorem 1.

Theorem 3. *Let* SchnorrSign *be a Schnorr signature scheme with κ-bit security and computational cost $T_{\mathcal{A}}$. Then the truncated signing algorithm as defined in Fig.2 where the first ℓ bits are fixed to string $s' \in \{0,1\}^\ell$, is also κ-bit secure.*

4 Truncating Primitives' Auxiliary Outputs

While in the previous section we showed how to truncate the primitive's main output, we also consider special cases where some primitives might include a secondary (or auxiliary) output. As an example, we consider the Winternitz one-time signature scheme (WOTS) [12], where the auxiliary output consists of a checksum of the number of zero bits, which is appended to the main output in order to prevent forgery (i.e. flipping 1-bits to 0's), as shown in Fig. 3. Here our technique can be applied by requiring a *fixed* checksum, therefore omitting it entirely from the signature. Because the checksum value has a much higher probability to fall within the median of the checksum range, we can require the fixed-value checksum to be simply the median of the range (as shown in Fig. 4), which provides a way of

[7] We note that one could create a *short* Schnorr signature, by assuming an $H()$ that maps to random λ-bit values, thus the final signature σ is of size 3λ-bits (2λ-bits to encode s and λ-bits to encode e) [11]. Our truncation technique can also apply on top of short Shnorr signatures.

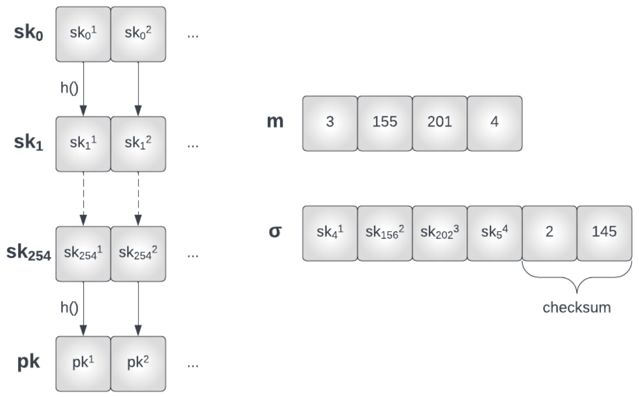

Fig. 3. Winternitz one-time signature example

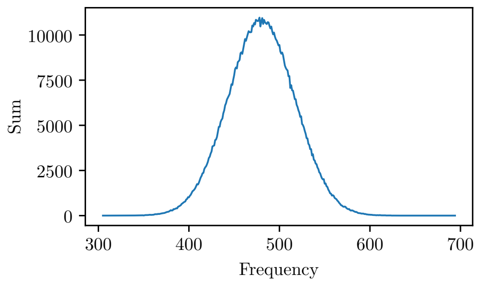

Fig. 4. WOTS checksum frequency diagram for $w = 16$ and 1 million signatures.

more efficient compression. Therefore, in these cases the computation cost paid upfront for truncating can be significantly lower compared to the "fixed-bit" approach discussed in Section 3. We showcase this efficiency benefit in Section 5. Note that a similar approach and implied security of constant sum WOTS has been discussed in [9].

5 Evaluation

We performed a series of evaluation experiments [13] to measure the trade-off between the truncation parameter and the computational effort for the primitives we considered. Our evaluation series were performed using fastcrypto library in Rust [15] on a Macbook M1 Pro as well as on an AWS t3.xlarge instance, using a single CPU core (note that our truncation algorithms are naturally parallelizable, but our implementation did not apply multi-threads focusing on the worst case).

We present our results for the primitives discussed in Section 3[8] in Table 1, using a sample size of 100 for one byte of truncation (ℓ=8). For more bytes there is an factor of 2^8 computational cost blowup for each additional byte truncated, therefore our results can be naturally extrapolated to derive the expected costs for larger truncation, as shown in Fig. 5. Consequently, the equilibrium between the tolerated computational overhead and the desired truncation benefits ultimately depends on the specific primitive and its application scenario (e.g. even a week's worth of computational work might be tolerable in order to reduce the public address size by 5 bytes in a blockchain application, where the benefit will be permanent). Note that there are additional techniques we can apply to speed up the computation stage, e.g. using pre-computed lookup tables for public key generation and perform elliptic curve additions rather multiplications. We also evaluate truncated hashed public keys as shown in Table 2 (which are a common practice to derive public addresses in cryptocurrencies such as Bitcoin).

	Ed25519 public keys	secp256k1 public keys	Ed25519 signature	ECDSA secp256k1 signature	SHA2-256	SHA3-256	BLAKE3-256
Macbook Pro M1 MAX	3.0589 ms	4.0909 ms	3.2539 ms	15.649 ms	63.033 µs	71.786 µs	36.335 µs
AWS t3.xlarge	5.5355 ms	7.4514 ms	5.5283 ms	30.529 ms	110.11 µs	166.18 µs	43.198 µs

Table 1. Evaluation on truncated primitives for ℓ=8.

[8] Ed25519 is a deterministic scheme, but we assume analogy with a randomized version, effectively representing Schnorr signature schemes in this list.

	Ed25519 + SHA2-256	Ed25519 + SHA3-256	Ed25519 + BLAKE3-256	ECDSA secp256k1 + SHA2-256	ECDSA secp256k1 + SHA3-256	ECDSA secp256k1 + BLAKE3-256
Macbook Pro M1 MAX	3.1746 ms	3.3702 ms	3.1199 ms	4.0191 ms	4.1623 ms	3.9293 ms
AWS t3.xlarge	5.4751 ms	5.4873 ms	5.6650 ms	7.0932 ms	7.0677 ms	7.4188 ms

Table 2. Hashed public key truncation for ℓ=8.

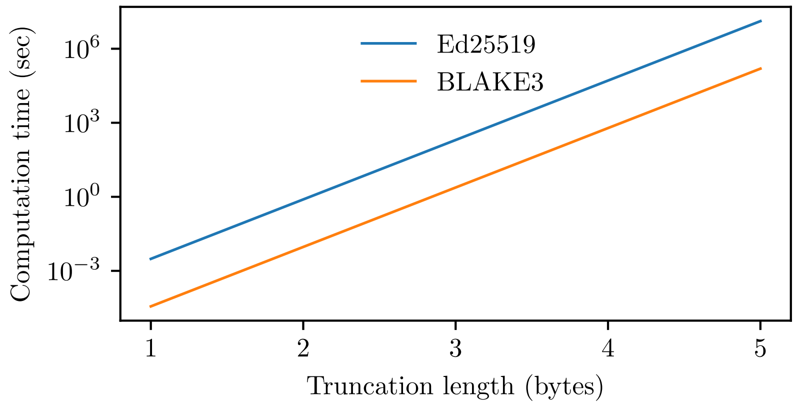

Fig. 5. Truncating primitives for larger truncation parameters

Truncated WOTS. We evaluate truncated WOTS separately, as it involves truncating its auxiliary output instead of truncating the main output in the other primitives we considered (i.e. requiring a fixed checksum instead of fixed bits). By considering the standard Winternitz parameter $w = 16$, the probability of successfully finding an output with the median checksum is roughly 1.1%. This implies that on average, truncated WOTS needs about 90 retries to output a valid one-time signature. Consequently, since WOTS is used in SPHINCS [10], the state-of-the-art stateless post-quantum signature scheme, with 64 hash elements output plus 3 elements as checksum, we can achieve about 4.5% compression with only $2^{6.5}$ effort (because of the bell-like normal distribution frequency curve in Fig. 4), compared to the “fixed-bit” approach used in truncated hash functions which would require 2^{12} effort to achieve the same level of compression. Note that we could also reduce the size of checksum instead of completely eliminating it (i.e. to 1 element instead of 3 for $w = 16$), which would require a lot less effort on the signer's side.

6 Conclusion and Future Directions

We presented Truncator, a paradigm to truncate the output size of cryptographic primitives with a computational trade-off. As a starting point, we showed how our approach can be applied on basic cryptographic primitives, while showing an additional benefit in certain types of primitives (e.g. in primitives with auxiliary outputs such as checksums). Our paradigm opens many possibilities for exploration, such as its implications in cryptoeconomics (e.g. the equilibrium of the trade-off when quantifying the benefits and the initial investment in computation), or the ways of applying it (e.g. delegating the computational effort to external services). We also intend to further explore how our truncation paradigm is applicable for more advanced cryptographic primitives (e.g. for truncated non-interactive zero-knowledge proofs) and formally prove their security. Finally, another future work direction is proposing novel primitives specifically handcrafted to utilize mining techniques on sender's side, towards improved efficiency for communication and receiver's work.

References

1. Ethereum gas golfing, https://ethereum.org/en/developers/docs/mev/#mev-extraction-gas-golfing
2. Evolution of the signature size in bitcoin, https://b10c.me/blog/006-evolution-of-the-bitcoin-signature-length/
3. Length of ecdsa signatures, https://transactionfee.info/charts/bitcoin-script-ecdsa-length/
4. Profanity ethereum vanity address tool, https://github.com/johguse/profanity
5. Vanitygen, https://en.bitcoin.it/wiki/Vanitygen
6. A vulnerability disclosed in profanity, an ethereum vanity address tool, https://blog.1inch.io/a-vulnerability-disclosed-in-profanity-an-ethereum-vanity-address-tool-68ed7455fc8c
7. Would it matter if my miner was hashing random vs incremental values?, https://crypto.stackexchange.com/questions/29318/would-it-matter-if-my-miner-was-hashing-random-vs-incremental-values
8. Random sampling vs incrementing randomness in cryptographic protocols (2021), https://crypto.stackexchange.com/questions/93651/random-sampling-vs-incrementing-randomness-in-cryptographic-protocols
9. Aumasson, J.P., et al.: Sphincs+ PQ NIST competition Round3 presentation, https://csrc.nist.gov/CSRC/media/Presentations/sphincs-round-3-presentation/images-media/session-1-sphincs-plus-hulsing.pdf
10. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: Practical stateless hash-based signatures. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 368–397. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_15
11. Blocki, J., Lee, S.: On the multi-user security of short schnorr signatures with preprocessing. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 614–643. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07085-3_21
12. Buchmann, J., Dahmen, E., Ereth, S., Hülsing, A., Rückert, M.: On the security of the winternitz one-time signature scheme. In: Nitaj, A., Pointcheval, D. (eds.) Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6737, pp. 363–378. Springer (2011). https://doi.org/10.1007/978-3-642-21969-6_23, https://doi.org/10.1007/978-3-642-21969-6_23
13. Chalkias, K., Chatzigiannis, P.: Rust truncator 0.1, https://github.com/MystenLabs/truncator/
14. Chalkias, K., Garillot, F., Kondi, Y., Nikolaenko, V.: Non-interactive half-aggregation of eddsa and variants of schnorr signatures. In: Paterson, K.G. (ed.) Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track. Springer (2021)
15. Chalkias, K., Garillot, F., Lindstrøm, J., Riva, B., Wang, J.: fastcrypto v0.1.3, https://crates.io/crates/fastcrypto
16. Dang, Q.: Nist special publication 800-106: Randomized hashing for digital signatures (2009), https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-106.pdf
17. Fleischhacker, N., Larsen, K.G., Simkin, M.: How to compress encrypted data. Cryptology ePrint Archive, Paper 2022/1413 (2022), https://eprint.iacr.org/2022/1413, https://eprint.iacr.org/2022/1413
18. McCurdy, W.: Hackers nab nearly 1m in crypto from ethereum vanity adress exploit, https://decrypt.co/110526/hackers-nab-nearly-1-million-crypto-ethereum-vanity-adress-exploit
19. Micciancio, D., Walter, M.: On the bit security of cryptographic primitives. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 3–28. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9_1
20. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2009), http://bitcoin.org/bitcoin.pdf
21. NIST: NIST key-length recommendations (2020), https://www.keylength.com/en/4/
22. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (May 1996). https://doi.org/10.1007/3-540-68339-9_33
23. Pornin, T.: Truncated EdDSA/ECDSA signatures. Cryptology ePrint Archive, Report 2022/938 (2022), https://eprint.iacr.org/2022/938
24. Seurin, Y.: On the exact security of Schnorr-type signatures in the random oracle model. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 554–571. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_33
25. Watanabe, S., Yasunaga, K.: Bit security as computational cost for winning games with high probability. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 161–188. Springer, Heidelberg (Dec 2021). https://doi.org/10.1007/978-3-030-92078-4_6