IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2020, No. S1, pp. 262–294. DOI:10.13154/tosc.v2020.iS1.262-294

The Subterranean 2.0 Cipher Suite

Joan Daemen[1], Pedro Maat Costa Massolino[1], Alireza Mehrdad[1] and Yann Rotella[2]

[1] Digital Security Group, Radboud University, Nijmegen, The Netherlands
[2] Laboratoire de Mathématiques de Versailles, University of Versailles Saint-Quentin-en-Yvelines (UVSQ), The French National Centre for Scientific Research (CNRS), Paris-Saclay University, Versailles, France
joan@cs.ru.nl, P.Massolino@cs.ru.nl, Alireza.Mehrdad@ru.nl, yann.rotella@uvsq.fr

Abstract. This paper presents the Subterranean 2.0 cipher suite that can be used for hashing, MAC computation, stream encryption and several types of authenticated encryption schemes. At its core it has a duplex object with a 257-bit state and a lightweight single-round permutation. This makes Subterranean 2.0 very well suited for low-area and low-energy implementations in dedicated hardware.

Keywords: lightweight · permutation-based crypto · deck function · XOF function · session authenticated encryption · NIST lightweight competition

1 Introduction

Subterranean is a cryptographic primitive to be used both for hashing and as a stream cipher and dates back to 1992 [CDGP93, Dae95]. With some imagination its mode can be seen as a precursor to the sponge [BDPA07] with an absorbing phase followed by a squeezing phase.

The round function of Subterranean has features that were adopted in several designs over the last three decades, including Panama [DC98], RadioGatún [BDPA06], Keccak-p [BDPA14] and Xoodoo [DHVAVK18]. Namely, all its steps, except the addition of a constant, are bit-level translation-invariant operations, its non-linear step is χ, the mixing step is a lightweight bit-oriented mapping with a heavy inverse and it has a bit shuffling step.

But the Subterranean round function also differs in important ways from Keccak-p and Xoodoo. Namely, its state is essentially one-dimensional rather than 3-dimensional, due to the particular shuffling and the 257-bit state, it is not software-friendly, and it has a buffer, similar to the *belt* in belt-and-mill designs such as RadioGatún [BDPA06].

Despite the differences, it only takes some minor refurbishing to turn Subterranean into a lightweight symmetric cipher suite that can compete with new designs, at least when implemented in dedicated hardware. Refurbishing we did, and we call the result Subterranean 2.0. In short, it is Subterranean with the buffer removed and the hashing and stream encryption modes replaced by a duplex-based construction with modes on top, inspired by the cyclist mode underlying Xoodyak [DHAK18]. The result is very efficient in hardware but remains relatively slow in software. We believe this makes sense in resource-constrained platforms, in particular, when energy per bit is the primary concern and relatively short messages must be protected. The current state of cryptanalysis of Subterranean and related designs suggests that it has a very good trade-off between safety margin and hardware performance.

Subterranean 2.0 operates on a state of 257 bits. The modernization into a duplex object required updating the output extraction and the input injection. For the former,

Received: 2019-12-10, Accepted: 2020-02-15, Published: 2020-06-22

we have opted to extract a 32-bit string z per duplex call, where each bit of z is the sum of 2 state bits. For the latter, we inject a string σ of up to 32 bits per duplex call in keyed mode and up to 8 bits every two rounds in unkeyed mode. The central functions of the duplex object are the application of a permutation to the state and subsequent injection of an input string on the one hand and extraction of an output string on the other. On top of this we define a number of wrapper functions for absorbing arbitrary-length strings, possibly combined with encryption or decryption, for performing blank rounds and for squeezing arbitrary-length strings.

Loyal to the original Subterranean, we chose for the permutation f in duplex to have only one round and so we expect there to be attacks better than generic ones, i.e., those not exploiting the specifics of f. In particular we claim 128 bits of security against multi-target attackers in keyed modes and 112 bits in unkeyed modes.

In Section 2 we specify the Subterranean duplex object, the primitive underlying the schemes we propose and the three cryptographic schemes that are specified as modes of on top of it: an eXtendable Output Function (XOF), a Doubly-Extendable Cryptographic Keyed (deck) function and a Session Authenticated Encryption (SAE) scheme. This is not meant to be exhaustive but covers most use cases: the XOF for hashing, the deck function for MAC computation, stream encryption, key derivation and more sophisticated modes such as those specified in the Xoodoo cookbook [DHAK18] and the SAE scheme for compact authenticated encryption. In Section 3 we provide the design rationale of the Subterranean 2.0 cipher suite and in Section 4 we discuss implementation aspects. Appendix A contains the details of a dedicated inner collision attack on a reduced-round version of Subterranean unkeyed hashing discussed in the paper, Appendix B lists examples of differential trails in Subterranean and Appendices C and D illustrate the basic operations and modes of Subterranean in figures. Finally, Appendix E illustrates a 4-round unrolled hardware architecture.

In the remainder of this paper we use the term *security strength* in relation to security claims and attacks. We say a cipher has security strength of s bits against an attack if the success probability p of an attack is at most 2^{-s} multiplied by the attack complexity, that is the sum of its data complexity and computational complexity.

2 Specification of the Subterranean 2.0 suite

We specify the Subterranean 2.0 suite in a bottom-up fashion, starting with the round function, input injection and output extraction in Section 2.1, the Subterranean 2.0 duplex object in Section 2.2, the XOF function in Section 2.3, the deck function in Section 2.4 and the SAE scheme in Section 2.5. Finally, Section 2.6 specifies the instances submitted to the NIST lightweight competition.

2.1 The round function R, input injection and output extraction

The round function R operates on a 257-bit state and has four steps:

$$\mathrm{R} = \pi \circ \theta \circ \iota \circ \chi \ . \tag{1}$$

Each step of the round function has a particular purpose: χ for non-linearity, ι for asymmetry, θ for mixing and π for dispersion. In the remainder we will use the term *bending* as a synonym for non-linearity.

We denote the state as s and its bits as s_i with position index i ranging from 0 to 256,

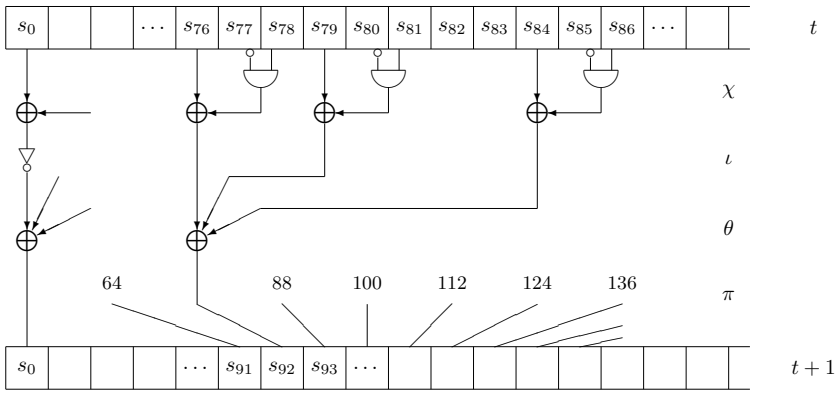

Figure 1: Subterranean round function, illustrated for bit s_{92}

where any expression in the index must be taken modulo 257. For all $0 \leq i < 257$:

$$\begin{aligned}
\chi: &\quad s_i \leftarrow s_i + (s_{i+1} + 1)s_{i+2} \ , \\
\iota: &\quad s_i \leftarrow s_i + \delta_i \ , \\
\theta: &\quad s_i \leftarrow s_i + s_{i+3} + s_{i+8} \ , \\
\pi: &\quad s_i \leftarrow s_{12i} \ .
\end{aligned}$$

Here the addition and multiplication of state bits are in $\mathbb{F}_2$, and δ_i is a Kronecker delta: $\delta_i = 1$ if $i = 0$ and 0 otherwise. We sometimes use subscript notation for indicating component functions of mappings, e.g. $\chi(s)_i$ denotes bit i of the state obtained by applying χ to s. Figure 1 illustrates the round function by the computational graph of a single bit of the state.

At the core of the Subterranean duplex object is a simple (internal) *duplex call* that first applies the Subterranean round function R to the state and then injects a string σ of variable length of at most 32 bits. Before adding it into the state, it pads the string σ to 33 bits with simple padding (10^*) and hence the absorbing rate is 33 bits. In between duplex calls, one may extract 32-bit strings z from the state, so the squeezing rate is 32 bits.

Each of the 32 bits of the extracted output z is constructed as the sum of two state bits. Those are taken from 64 fixed positions that are the elements of $\langle 12^4 \rangle$, the multiplicative subgroup of $(\mathbb{Z}/257\mathbb{Z})^*$ generated by $12^4 = 176$, that has order $\mathsf{ord}(176) = 64$. More precisely,

$$\text{for all } 0 \leq i \leq 31 : z_i \ = \ s_{12^{4i}} \ + \ s_{-12^{4i}} \ ,$$

where the minus sign is taken modulo 257, e.g., $-12^0 = -1 = 256$ and $-12^4 = -176 = 81$.

The 33 bits of σ after padding are injected into the state at positions that form the first 33 powers of 12^4 in $\langle 12^4 \rangle$. For the unkeyed mode (hashing), the input σ is limited to 8 bits bringing the effective absorbing rate to 9 bits due to the padding.

2.2 The Subterranean duplex object

The Subterranean duplex object has at its core two internal functions: the duplex call and the output extraction. The duplex call applies the round function and injects the input together with the output extraction it is specified in the previous section.

On top of the duplex and extraction calls it has a thin wrapper consisting of three functions that facilitate the compact specification of cryptographic functions and schemes.

The wrapper supports absorbing and squeezing of strings of arbitrary length and the integration of encryption and decryption with absorbing. It provides separators between

absorbed strings by internally ensuring its last injected block to be shorter (possibly empty) than 32 bits in keyed mode and 8 bits in unkeyed mode. When absorbing into a *secret* state (op $\in$ {keyed, encrypt, decrypt} in Algorithm 1), Subterranean injects up to 32 bits per duplex call. In unkeyed absorbing (op = unkeyed), it injects at most 8 bits per duplex call and applies duplex calls with no injected data in between. We motivate this choice with cryptanalysis that we present in Section 3.4.

We make no security claim for the Subterranean duplex object as such but only for the schemes that consist of modes on top of the primitive.

We specify the Subterranean duplex object in Algorithm 1. There, any input or output is a bit string of arbitrary length, unless specified otherwise. We indicate the length of a bit string X by $|X|$, concatenation of strings a and b by $a||b$ and the empty string by ϵ.

Algorithm 1 Subterranean duplex object

Interface: Constructor: Subterranean()
  $s \leftarrow 0^{257}$

Interface: $Y \leftarrow$ absorb(X, op) with op $\in$ {unkeyed, keyed, encrypt, decrypt}
  if op = unkeyed **then** $w \leftarrow 8$ **else** $w \leftarrow 32$
  Let $x[n]$ be X split in w-bit blocks, with last block strictly shorter
  $Y \leftarrow \epsilon$
  for $i = 0, \ldots, n-1$ **do**
    if op $\in$ {encrypt, decrypt} **then**
      temp $\leftarrow x[i]$ + (extract(s) truncated to |x[i]|)
      $Y \leftarrow Y||$temp
    if op = decrypt **then** duplex(temp) **else** duplex$(x[i])$
    if op = unkeyed **then** duplex(ϵ)
  return Y

Interface: blank(r) with r a natural number
  for r times **do** duplex(ϵ)

Interface: $Z \leftarrow$ squeeze(ℓ) with ℓ a natural number
  $Z \leftarrow \epsilon$
  while $|Z| < \ell$ **do**
    $Z \leftarrow Z||$extract(s)
    duplex(ϵ)
  return Z truncated to ℓ bytes

Internal interface: duplex(σ) with $|\sigma| \leq 32$
  $s \leftarrow \mathrm{R}(s)$
  $x \leftarrow \sigma||1||0^{32-|\sigma|}$
  for j from 0 to 32 **do** $s_{12^{4j}} \leftarrow s_{12^{4j}} + x_j$

Internal interface: $z \leftarrow$ extract(s)
  $z \leftarrow \epsilon$
  for j from 0 to 31 **do** $z \leftarrow z||(s_{12^{4j}} + s_{-12^{4j}})$
  return z

2.3 The Subterranean-XOF function

We specify Subterranean-XOF in Algorithm 2. It is meant to be used for unkeyed hashing and takes as input a sequence of an arbitrary number of arbitrary-length strings $M[i]$,

denoted as $M[[n]]$ and returns a bit string of arbitrary length. For Subterranean-XOF we make a flat sponge claim.

Claim 1 (Flat sponge claim [BDPA11a])**.** *The success probability of any attack on Subterranean-XOF shall not be higher than the sum of that for a random oracle and*

$$\frac{N^2}{2^{224}}$$

with N the attack complexity expressed in calls to the Subterranean round function. We exclude from the claim weaknesses due to the mere fact that the function can be described compactly and can be efficiently executed, e.g., the so-called random oracle implementation impossibility [MRH04], as well as properties that cannot be modeled as a single-stage game [RSS11].

Our claim corresponds to a security strength of 112 bits against all attacks that do not apply to a random oracle. The capacity in the claim is 24 bits short of the effective capacity $257 - 9 = 248$ bits to account for possible shortcut attacks. These are attacks that are more efficient than generic ones by exploiting specific properties of Subterranean (see Section 3.4).

Algorithm 2 Subterranean-XOF

Interface: $Z \leftarrow$ Subterranean-XOF$(M[[n]], \ell)$ with $M[[n]]$ a string sequence and ℓ a natural number
$S \leftarrow$ Subterranean()
for all strings $M[i]$ in $M[[n]]$ **do** S.absorb$(M[i], \text{unkeyed})$
S.blank(8)
return $Z \leftarrow S$.squeeze(ℓ)

2.4 The Subterranean-deck function

We specify Subterranean-deck in Algorithm 3. It takes as input an arbitrary-length key K and a sequence of an arbitrary number of arbitrary-length strings $M[i]$, denoted as $M[[n]]$ and returns a bit string of arbitrary length. It can readily be used as a stream cipher, a MAC function and for key derivation. The Farfalle paper [BDH+17] and the Xoodoo cookbook [DHAK18] specify several authenticated encryption modes for deck functions.

We claim Subterranean-deck offers 128 bits of security strength against adversaries that are limited to 2^{96} data blocks, when it is loaded with a key with min-entropy 128 bits in a single-target setting, and loaded with independent keys with min-entropy $128 + \log_2 \mu$ in a multi-target setting with μ targets.

Subterranean-deck absorbs the key in blocks of 32 bits. This makes it an application of the result of Bart Mennink [Men18]. The bottom line is that, even for a uniform key with length k and an ideal underlying permutation, the success probability of key prediction cannot be proven to be close to $N2^{-k}$ for N operations. On the other hand, the absence of this bound does not imply there is an attack with success probability above $N2^{-k}$ and we do not take this into account in our claim.

Claim 2. *The advantage of an adversary in distinguishing an array of μ instances of Subterranean-deck loaded with μ independent keys, each with min-entropy $128 + \log_2 \mu$ bits, from an array of μ independent random oracles, is upper bounded by $(N + M)2^{-128}$, with N the total computational complexity in calls to the Subterranean round function and M the total data complexity in 32-bit input- and output blocks, with $M < 2^{96}$.*

Algorithm 3 Subterranean-deck

Interface: $Z \leftarrow$ Subterranean-deck$(K, M[[n]], \ell)$ with $M[[n]]$ a string sequence and ℓ a natural number
$S \leftarrow$ Subterranean()
S.absorb$(K,$ keyed$)$
for all strings $M[i]$ in $M[[n]]$ **do** S.absorb$(M,$ keyed$)$
S.blank(8)
return $Z \leftarrow S$.squeeze(ℓ)

2.5 The Subterranean-SAE authenticated encryption scheme

We specify Subterranean-SAE in Algorithm 4. It takes a nonce when starting the session and can then encipher and authenticate a sequence of messages each consisting of a plaintext and associated data. Compared to authenticated encryption modes based on Subterranean-deck, Subterranean-SAE has smaller state and is better suited to offer protection against differential power analysis (DPA). In particular, the security is based on the secrecy of a state that evolves during the session rather than a static key. Across sessions, one can derive a fresh key per session using Subterranean-deck. This protects against differential power analysis and provides fine-grained forward secrecy. If one wishes to use the same key for multiple sessions and DPA is a concern, one can absorb the nonce bit per bit, as was proposed by Mostafa Taha and Patrick Schaumont in [TS14]. By taking as nonce the shortest binary representation of a session counter, this can be quite economical.

For Subterranean-SAE, we make the same security claim as for Subterranean-deck with two differences. First, we do not try to distinguish it from a random oracle, but from a random function with the same interface as Subterranean-SAE. Second, we only consider adversaries that respect nonces and that only get an error message when presenting invalid cryptograms for unwrapping.

Claim 3. *Consider an adversary that respects the nonce requirement and that, when presenting invalid cryptograms for unwrapping, only gets an error message. The advantage of such an adversary in distinguishing an array of μ Subterranean-SAE instances loaded with μ independent keys, each with min-entropy $128 + \log_2 \mu$ bits, from an array of μ independent random functions with the same interface is upper bound by $(N + M)2^{-128}$, with N the total computational complexity in calls to the Subterranean round function and M the total data complexity in 32-bit input and output blocks, with $M \leq 2^{96}$.*

This claim implies that, in the case of a 128-bit tag, the probability that an attacker violates plaintext confidentiality and the probability of cryptogram forgery are both $(N + M + q)2^{-128}$ with q the number of decryption queries. In other words, it implies in plaintext confidentiality and message integrity of security strength 128 bits.

In a nonce-misuse scenario or when unwrapping invalid cryptograms returns more information than a simple error, we make no security claims and the attacker may be able to recover the secret state. Fukang Liu, Takanori Isobe and Willi Meier showed an attack achieving exactly this, requiring 1177 sessions with the same diversifier and exchanging 224 bits in each, using conditional cube testers and guess-and-determine techniques in [LIM19].

2.6 Parameter choices for the NIST lightweight competition

In this section we specify the set of parameters for both the hash function and the authenticated encryption scheme in the context of the NIST lightweight competition. Both concrete instances are modes on top of the Subterranean duplex object that can be shared.

The hash function of the Subterranean 2.0 suite submitted to the NIST lightweight competition is Subterranean-XOF, with the input restricted to a string sequence of a single

Algorithm 4 Subterranean-SAE, with τ the tag length

Interface: start(K, N)
  $S \leftarrow$ Subterranean()
  S.absorb$(K,$ keyed$)$
  S.absorb$(N,$ keyed$)$
  S.blank(8)

Interface: $(Y, T) \leftarrow$ wrap(A, X, T', op) with op $\in \{\text{encrypt}, \text{decrypt}\}$
  S.absorb$(A,$ keyed$)$
  $Y \leftarrow S$.absorb(X, op)
  S.blank(8)
  $T \leftarrow S$.squeeze(τ)
  if op $=$ decrypt AND $(T' \neq T)$ **then** $(Y, T) \leftarrow (\epsilon, \epsilon)$
  return (Y, T)

arbitrary-length byte string and the output length fixed to 256 bits. The security claim is given in Claim 1

The authenticated encryption scheme of the Subterranean 2.0 suite submitted to the NIST lightweight competition is Subterranean-SAE, with a key K of length 128 bits, a nonce N of length 128 bits and a tag length $\tau = 128$ and associated data and plaintext limited to byte strings. The security claim is given in Claim 3 and it implies that the amount of data available to the attacker shall be limited to 2^{96} bits.

3 Design Rationale

In this section we give the design rationale for the Subterranean 2.0 cipher suite, following the canvas of the specifications in previous section. Before discussing the different aspects more in-depth, we give the design rationale in a nutshell.

3.1 Rationale in a nutshell

The round function operates at bit level with a maximum of symmetry by using very light translation-invariant operations and a minimum of sub-structures by letting it operate on a prime-sized state. When speaking of sub-structures, we are thinking of bytes and super-boxes in Rijndael [DR02], but also Matryoshka in Keccak-p [BDPA11b] or symmetry properties in the ChaCha permutation [Ber08, FLM10]. In a way, the Subterranean round function is the nec plus ultra of weak alignment.

We extract the output z from state bits that are in positions distant from each other to make it very hard to reconstruct the secret state from a series of outputs z and to prevent existence of measurable biases in the output stream Z.

Likewise, we inject input strings σ in the state in positions distant from each other to make it infeasible to control difference propagation in the state. In unkeyed absorbing we limit the strings σ in length to 8 bits and we apply two rounds in between input injections. The reason for this is to make the generation of state collisions infeasible.

Subterranean-XOF and Subterranean-deck each have a single absorbing phase followed by a squeezing phase. In between those phases there is a sequence of 8 blank rounds. In Subterranean-deck, these blank rounds are meant to prevent measurable correlations or exploitable differentials between input $M[i]$ and output Z. In Subterranean-XOF they are meant to make the generation of collisions in n-bit outputs, for any $n \leq 224$, infeasible. Similarly, they should prevent the generation of (2nd) pre-images for any n-bit output with $n \leq 112$ in less than 2^n operations.

More generally, in Subterranean-XOF, Subterranean-deck and Subterranean-SAE alike, the blank rounds should make the output Z (tag or keystream) depend on all bits of the input in a complex way, as in the case for a random oracle. Finally, the blank rounds should prevent attacks that make use of higher order differentials such as cube attacks by the fact that expressions of state bits as a function of the state 8 rounds ago has high degree and is relatively dense.

The width of the permutation, 257, fits nicely the ambition to offer a security strength of 128 bits in keyed modes and 112 bits in unkeyed mode. It is rather small but not too small to fall prey to time-memory-data-precomputation trade-offs.

3.2 The round function

The round function is just taken from the original Subterranean specified in [Dae95], with the buffer addition removed and the bending step complemented. In short, it is a classical lightweight bit-oriented wide trail design, with a bending layer, a mixing layer, a shuffling layer and a (round) constant addition. The state is a one-dimensional array of length 257, a prime number. This had to be at least 256 as the original Subterranean targeted a security strength of 128 bits. A prime was taken to rule out exploitable symmetries. In particular, the bending and mixing layers are translation-invariant. The shuffling layer, that we call *multiplicative* as it moves each bit to a position that is a modular multiple of its original position, satisfies a generalization of translation-invariance. The round constant addition ensures the round function itself does not have any of those symmetry properties.

In the remainder of this section we discuss the propagation properties of the steps of the round function with an emphasis on their symmetry properties and the consequences for their algebraic order.

3.2.1 On translation-invariance

Definition 1 (Translation Invariance)**.** We say a mapping α is *translation-invariant* over an offset v if it commutes with a (cyclic) translation by v positions, that we denote as $\tau_{(v)}$:

$$\alpha \circ \tau_{(v)} = \tau_{(v)} \circ \alpha \ .$$

Lemma 1. *If a mapping is translation-invariant over offset 1, it is translation-invariant over any offset.*

Proof. We have: $\alpha \circ \tau_{(v)} = \alpha \circ \tau_{(1)} \circ \tau_{(v-1)} = \tau_{(1)} \circ \alpha \circ \circ \tau_{(v-1)}$. By iteratively applying this, we obtain $\tau_{(v)} \circ \alpha$. □

Lemma 2. *The composition of two translation-invariant mappings is translation-invariant.*

Proof. We have

$$\tau_{(1)} \circ \alpha \circ \beta = \alpha \circ \tau_{(1)} \circ \beta = \alpha \circ \beta \circ \tau_{(1)}$$

□

A translation-invariant mapping α is fully specified by its first component function $\alpha(s)_0$. Any other component function is given by $\alpha(s)_i = \alpha(\tau_{(i)}(s))_0$.

The shuffling layer π is not translation-invariant but has a similar property.

Definition 2 (Multiplicative shuffle)**.** Let b a strictly positive integer and g coprime to b. Then the mapping $\pi_{(g)} : s \in \mathbb{F}_2^b \to \mathbb{F}_2^b$ defined by $\forall 0 \leq j < b$,

$$\pi_{(g)}(s)_{gj \bmod b} = s_j$$

is called *multiplicative shuffle* on b-bit with shuffling factor g.

Lemma 3. *Let $\pi_{(g)}$ be a multiplicative shuffle, then $\tau_{(g)} \circ \pi_{(g)} = \pi_{(g)} \circ \tau_{(1)}$.*

Proof. Let $0 \leq j < b$ and $s \in \mathbb{F}_2^b$. Then $(\tau_{(g)} \circ \pi_{(g)}(s))_j = (\pi_{(g)}(s))_{j-g \bmod b}$. As $(\pi_{(g)}(s))_{gj \bmod b} = s_j$ and g is coprime to b, it implies that $(\pi_{(g)}(s))_{j-g \bmod b} = s_{g^{-1}(j-g)}$ where g^{-1} denotes the multiplicative inverse of g in $\mathbb{Z}/b\mathbb{Z}^*$. Hence $(\tau_{(g)} \circ \pi_{(g)}(s))_j = s_{g^{-1}j-1}$.

Similarly, we have $(\pi_{(g)} \circ \tau_{(1)}(s))_j = (\pi_{(g)}(\tau_{(1)}(s)))_j = (\tau_{(1)}(s))_{g^{-1}j}$. Hence $(\pi_{(g)} \circ \tau_{(1)}(s))_j = s_{g^{-1}j-1}$. □

3.2.2 The bending layer χ

The bending layer is χ, described and analyzed in [Dae95] and well known from Keccak-p. χ is the sparsest translation-invariant mapping of algebraic degree 2 that is invertible when the state has odd length. By sparse we mean that each output bit only depends on few input bits, in this case 3 bits in neighbouring positions. The low degree is an advantage when countermeasures against differential power analysis need to be implemented, such as masking or threshold schemes. In general, computing the inverse of χ requires a recursive procedure and the consequence is that it is dense and has an algebraic degree that is proportionate to the state length. In the case of Subterranean, the inverse of χ has algebraic degree 128. This complexity helps in frustrating cryptanalysts.

3.2.3 The mixing layer θ

The mixing layer θ is similarly sparse: each output bit is the sum of 3 input bits at fixed relative offsets. The Subterranean round function was an improved version of that of the very first wide-trail design, the hash function Cellhash [DGV91]. In Cellhash, the offsets in θ were $-3, 0, 3$: symmetric around 0 and at minimum distance so that each output bit of $\theta \circ \chi$ depends on 9 bits. Due to this choice of offsets, an input difference in θ with two active bits in positions $i + 3$ and $i + 6$ for any i leads to a 2-bit output difference with active bits in positions i and $i + 9$. The choice of offsets in Subterranean avoids this problem: a 2-bit input difference leads to an output difference with at least 4 active bits. The symmetry around 0 was abandoned, 3 was kept and 8 was chosen as the smallest value that gives excellent mixing properties.

In order to capture algebraic properties of θ, we identify state values s with binary polynomials $s(X) = \sum_i s_i X^i$. As discussed in [Dae95], a translation-invariant mapping corresponds with multiplication with a polynomial modulo $1 + X^b$. For a linear translation-invariant mapping α defined by $\alpha(s)_0 = \sum_i p_i s_i$, the mapping corresponds with multiplying with polynomial $P(X) = \sum_i p_i X^{b-i}$. For θ this yields:

$$\theta\left(s(X)\right) = s(X)(1 + X^{249} + X^{254}) \bmod (1 + X^{257})\,.$$

The multiplication polynomial can likewise be written as $1 + X^{-3} + X^{-8}$.

The polynomials $P(X)$ of degree smaller than 257 that are coprime to $1 + X^{257}$ form a group that we will denote by $\left(\mathbb{F}_2(X)/(1 + X^{257})\right)^*$. The modulus $1 + X^{257}$ is the product of $1 + X$ with 16 irreducible polynomials of degree 16. This implies that $\left(\mathbb{F}_2(X)/(1 + X^{257})\right)^*$ is isomorphic to the product of 16 groups of order $2^{16} - 1$ and $\mathsf{ord}(\theta)$ must divide $2^{16} - 1 = 3 \times 5 \times 17 \times 257$. For $P(X) = 1 + X^{-3} + X^{-8}$, we checked that the order is $2^{16} - 1$, the maximum possible order in $\left(\mathbb{F}_2(X)/(1 + X^{257})\right)^*$. The inverse of θ can be computed with the extended Euclidean algorithm or as $(1 + X^{-3} + X^{-8})^{2^{16}-2}$ and has a Hamming weight of 127. This high diffusion in the backward direction helps in frustrating cryptanalysis.

3.2.4 The shuffling layer π

The shuffling layer of Subterranean is a multiplicative shuffle (see definition 2) on 257 bits with shuffling factor 150. It puts bits that are 12 positions apart next to each other:

$s_i \leftarrow s_{12i}$. This ensures that each state bit depends on 81 bits of the state 2 cycles ago. Moreover, it moves neighbouring bits to positions 150 bits apart: $s_{150j} \leftarrow s_j$ as $150 \times 12 \bmod 257 = 1$. The result is that a single-bit difference in the state may affect 81 state bits 2 cycles later.

The order of 12 in $(\mathbb{Z}/257\mathbb{Z})^*$ is 256, or in other words, it is a generator. The consequence is that the order of π is likewise 256.

3.2.5 The order of the linear layer $\pi \circ \theta$

For understanding the order of the linear layer $\pi \circ \theta$ we first prove a number of lemmas.

Lemma 4. *If α is translation-invariant and $\pi_{(g)}$ a multiplicative shuffle, then $\pi_{(g)}^{-1} \circ \alpha \circ \pi_{(g)}$ is translation-invariant.*

Proof. We have:

$$\left(\pi_{(g)}^{-1} \circ \alpha \circ \pi_{(g)}\right) \circ \tau_{(1)} = \pi_{(g)}^{-1} \circ \alpha \circ \tau_{(g)} \circ \pi_{(g)} = \pi_{(g)}^{-1} \circ \tau_{(g)} \circ \alpha \circ \pi_{(g)} = \tau_{(1)} \circ \left(\pi_{(g)}^{-1} \circ \alpha \circ \pi_{(g)}\right)$$

□

Lemma 5. *If α is translation-invariant and $\pi_{(g)}$ a multiplicative shuffle and $\beta = \pi_{(g)}^{-1} \circ \alpha \circ \pi_{(g)}$, then $\beta(s)_0 = \alpha(\pi_{(g)}(s))_0$.*

Proof. Bit 0 of $\pi_{(g)}^{-1}(\alpha(\pi_{(g)}(s)))$ is bit 0 of $\alpha(\pi_{(g)}(s))$ and that is per definition $\alpha(\pi_{(g)}(s))_0$. □

Lemma 6. *If α is linear and translation-invariant with multiplication polynomial $P(X)$ and $\pi_{(g)}$ a multiplicative shuffle, then $\pi_{(g)}^{-1} \circ \alpha \circ \pi_{(g)}$ is linear and translation-invariant with multiplication polynomial $P(X^g)$.*

Proof. Applying Lemma 5 with $\alpha = \theta$ we obtain $\theta^{(j)}(s)_0 = \theta(\pi_{(g)}(s))_0 = \sum_i p_i s_{ig}$. This corresponds with polynomial $P(X^g)$. □

Lemma 7. *For any $i \in \mathbb{Z}$, let $\theta^{(i)}$ be the linear transformation defined as $\theta^{(i)} = \pi^{-i} \circ \theta \circ \pi^i$. Then, for any $n \in \mathbb{N}$, we have:*

$$(\pi \circ \theta)^n = \pi^n \circ \theta^{(n-1)} \circ \theta^{(n-2)} \circ \cdots \circ \theta^{(1)} \circ \theta^{(0)} \ . \tag{2}$$

Proof. Working it out gives:

$$(\pi \circ \theta)^n = \pi \circ \theta \circ \pi \circ \theta \circ \cdots \circ \pi \circ \theta = \pi^n \circ \pi^{1-n} \circ \theta \circ \pi^{n-1} \circ \pi^{2-n} \circ \theta \circ \cdots \circ \pi^1 \circ \theta \ .$$

□

If we apply Lemma 7 for $n = 256$, π^n becomes the identity and we obtain a mapping that is an element of $\left(\mathbb{F}_2(X)/(1 + X^{257})\right)^*$ as it is the composition of 256 linear translation-invariant mappings, each with its own multiplication polynomial. From this follows that the composed mapping is also a linear translation-invariant mapping and that its order divides $2^{16} - 1$. Hence, the order of $\pi \circ \theta$ divides $2^8(2^{16} - 1)$. We checked the divisors of this integer and it turns out that the order of $\pi \circ \theta$ is 256 and the minimal polynomial of $\pi \circ \theta$ is $1 + X^{256}$.

More in general, we can prove the following lemma.

Lemma 8. *Let θ' be a linear translation-invariant mapping with multiplication polynomial P and let $\pi' = \pi_{(g)}$ a 257-bit multiplicative shuffle. If $\mathsf{ord}(g) \geq 16$, the order of $\pi' \circ \theta'$ divides $\mathsf{ord}(g)$.*

Proof. Applying (2) with $n = \mathsf{ord}(g)$ yields

$$(\pi' \circ \theta')^{\mathsf{ord}(g)} = \theta'^{(\mathsf{ord}(g)-1)} \circ \theta'^{(\mathsf{ord}(g)-2)} \circ \cdots \circ \theta'^{(1)} \circ \theta'^{(0)}$$

with $\theta'^{(i)} = \pi'^{-i} \circ \theta' \circ \pi'^{i}$. For any $i \in \mathbb{N}$, the multiplication polynomial of $\theta'^{(i)}$ is $P(X^{g^i})$ modulo $1 + X^{257}$ (see Section 4.4).

It follows that $(\pi' \circ \theta')^{\mathsf{ord}(g)}$ is a linear translation-invariant mapping with polynomial $Q = \prod_{i=0}^{\mathsf{ord}(g)-1} P(X^{g^i})$. As expressed before in Section 3.2.3, $1 + X^{257}$ is the product of $1 + X$ and 16 polynomials of degree 16. We can now show that $Q = 1$, the multiplication polynomial of the identity mapping. Let α be a primitive element of $\mathbb{F}_{2^{257}}$, then it follows immediately from the expression of Q that $Q(\alpha) = Q(\alpha^g) = Q(\alpha^{g^2}) = \cdots = Q(\alpha^{g^{\mathsf{ord}(g)-1}})$. As α is a primitive element, all these powers are different elements. Thanks to the Chinese Remainder Theorem, and the factorisation of $(1 + X^{257})$ in polynomials of degree 16 or smaller, the result follows immediately: Q is a constant polynomial. Depending on the Hamming weight of P, we get either a constant term that is zero or one, but as P corresponds to a bijective mapping, we have $Q = 1$. Hence, $(\pi' \circ \theta')^{\mathsf{ord}(g)} = \mathsf{Id}$ if $\mathsf{ord}(g) \geq 16$. □

3.3 The number of blank rounds

The algebraic degree of the Subterranean round function is 2 and for small r the degree of r rounds is 2^r. For larger r, it was observed by Christina Boura, Anne Canteaut and Christophe De Cannière in [BCC11] that the degree lags behind starting from a number of rounds r when 2^r starts approaching the permutation width b. Still, for $r = 8$ we expect the algebraic degree to be well above 128. Together with the positioning of the input bits and the fact that there are only 32 per round, this makes it unlikely that shortcut attacks based on higher-order differentials, such as cube attacks, can be mounted.

In collision attacks in Subterranean-XOF, a non-zero difference a' in the state before the blank rounds will propagate to a non-zero difference b' in the state after the blank rounds. Due to the 8 rounds, the difference b' will depend in a complicated way on the difference a' and the absolute values of the states before the blank rounds. This implies that the best collision-generating strategy against Subterranean-XOF is to go for an internal collision during the absorbing phase.

For Subterranean-deck, differential attacks require the propagation of a difference across the blank rounds. As can be seen in Section 3.7, we did not find 8-round trails and we are certain that there are none with expected differential probability above 2^{-98}. It follows that exploiting differential propagation with 2^{96} or less pairs will be problematic. Similarly, we do not expect there to be 8-round correlations with amplitude above 2^{-48}, the value that would be required for exploiting it with 2^{96} input-output couples. The same reasoning is true for Subterranean-SAE for nonce-respecting adversaries that do not get unverified deciphered ciphertext. Namely, in these use cases, controllable input and output is separated by 8 blank rounds.

The safety margin provided by the 8 blank rounds in keyed modes is confirmed by the cryptanalysis of Fukang Liu, Takanori Isobe and Willi Meier in [LIM19]. The authors showed that when the number of blank rounds is reduced to 4, one can do a state recovery attack with data complexity $2^{71.5}$ bytes and computational complexity 2^{122} computations of the round function.

In the following sections we will discuss input-only attacks in the form of state collisions and output-only attacks in the form of state-recovery attacks from output only and biases in the output.

3.4 State collisions in unkeyed absorbing

The single-most important security requirement of unkeyed absorbing is that it should be hard to generate state collisions. A state collision is defined as a pair of different string sequences $M[[n]]$ and $M'[[n']]$ that lead to the same state value.

A string sequence $M[[n]]$ gives rise to a sequence of absorb calls, that each split the string $M[i]$ into 8-bit blocks, pad these blocks with 10^* and then sequentially inject these into the state in a series of duplex calls interleaved with blank duplex calls. The borders between the strings in the sequence are marked by the fact that the last block of each string $M[i]$ before padding is shorter than 8 bits. If the strings $M[i]$ are not restricted to byte strings but can have any bitlength, an adversary can hence choose 9 bits between two consecutive duplex calls. In the case of input byte strings, the adversary can choose from $2^8 + 1$ input values, namely 2^8 byte values with padding and the padded empty string. This reduces the number of chosen bits per two consecutive duplex calls to $8 + \epsilon$ with ϵ small. Our ambition is to have 112-bit security strength in the more general case of bit strings.

When generating a state collision, the colliding string sequences $M[[n]]$ and $M'[[n']]$ may take an equal or a different number of duplex calls. For duplex call sequences of equal length, we can try to find a differential from the input $M[[n]]$ to a zero difference in the state with a high differential probability (DP). For duplex call sequences of different lengths, one could try to generate a *fixed point*. Those properties are discussed respectively in Section 3.4.4 and Section 3.4.3. Finally, one can try to find state collisions with a generic attack by just randomly trying inputs and count on the birthday bound for collisions to occur. This is the starting point of the attack explained in the following subsections.

3.4.1 Generating state collisions with a generic attack

In absorbing, we call the bit positions where the input is injected the *outer part* of the state. The *inner part* of the state is formed by the other bit positions. In unkeyed absorbing, the outer part consists of 9 bit positions and the inner part 248 bit positions. In keyed absorbing, the outer part is 33 bits wide and the inner part 224 bits.

A naive collision-generating attack strategy is to try random inputs and to check if there are state collisions among them. The probability that there is a collision after q attempts is close to $\binom{q}{2}/2^{257} \approx q^2/2^{258}$ as long as q is smaller than 2^{129}. When q gets close to this value, it is likely that the set of inputs exhibits a state collision. The success probability increases if we only require a collision in the 248-bit inner part of the state, a so-called *inner collision*. An inner collision can readily be converted into a state collision by compensating for the (possible) difference in the 9-bit outer part by choosing the last blocks in the inner-state colliding inputs. The probability of an inner collision after q attempts is about $q^2/2^{249}$. As every attempt will take at least two duplex calls, the lowest possible computational complexity of q inputs is $N = 2q$. It follows that the probability of an inner collision for this generic attack is close to $N^2/2^{251}$.

Increasing this probability by a factor higher than $2^{251-222} = 2^{29}$ by exploiting specifics of the round function would break our security claim for Subterranean-XOF, Claim 1 as a state collision allows generating an arbitrary number of input pairs with the same output, something that a random oracle does not have.

3.4.2 Dedicated attack to generate inner collisions

In this section we describe an attack for generating an inner collision for a round-reduced version of Subterranean-XOF. It is round-reduced in the sense that there are no blank duplex calls in between the duplex calls absorbing the input blocks. As a matter of fact, this attack is the reason why we introduced these blank duplex calls in unkeyed absorbing.

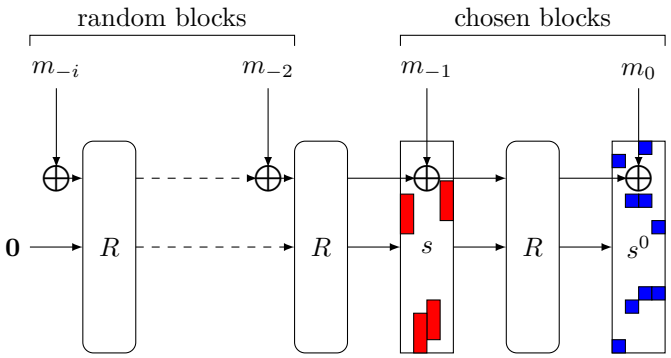

Figure 2: Finding state collisions in unkeyed absorbing. The m_i's are 9-bit padded input blocks and we aim for a collision on s^0. We tolerate a difference in s^0 the blue positions as it can be compensated by a simple difference in m_0. Red positions in s represent conditions that can be satisfied by choosing the value of m_{-1},

An inner collision can be seen as a stepping stone to a state collision, where the difference in the outer part can be annihilated by compensating last input blocks. The states s and s' corresponding to the inner collision are equal in their outer part.

The basic idea of this attack is to take this one step further. We build *proto-collisions*: pairs of states s and s' that satisfy specific equations. For states of a proto-collision, there is a high probability that their difference in the inner part can be annihilated by compensating last input blocks.

In a first phase of the attack, we generate states s by absorbing random inputs and assemble them in a set we call *birthday set*.

In a second phase of the attack, we identify pairs of states (s, s') in this set that form a proto-collision. We depict the schematic of the attack in Figure 2.

We now derive the equations that a proto-collision must satisfy. We denote by s and s' the value of the state just before absorbing the penultimate input blocks m_{-1} and m'_{-1}, that we denote by b and b' for compactness. Our equations are in the bits of s, s', b and b' as variables, i.e., s_i and s'_i for $0 \leq i < 256$ and b_i and b'_i for $0 \leq i < 9$. We aim to find equations in s and s' such that if satisfied, the probability that they can be converted into an inner collision by choosing appropriate last blocks b and b' is maximal. The states just before absorbing m_0 and m'_0 form an inner collision if their difference is contained in the 9-bit outer part, i.e., in bit positions $1, 176, 136, 35, 249, 134, 197, 234, 64$, highlighted in blue in Figure 2. For the 248 inner bits, the difference must be 0 and one can express the difference in terms of the bits of s, s', b and b'. As the round function has algebraic degree 2, these are quadratic equations. We denote these equations as:

$$q_j(s, b) = q_j(s', b') \text{ for } 1 \leq j \leq 248 \,,$$

with q_j quadratic expressions. Each bit b_i (or b'_i), for $0 \leq i < 9$ appears in the expression q_j in 9 of those equations. By doing a Gaussian elimination, we can convert this to an equivalent system in which the number of equations where b_i or b'_i appear is minimized. In particular, it contains 221 equations where the variables b_i and b'_i don't intervene at all. We denote the resulting system as:

$$q'_j(s, b) = q'_j(s', b') \text{ for } 1 \leq j \leq 26 \tag{3}$$

$$q'_j(s) = q'_j(s') \text{ for } 27 \leq j \leq 248 \tag{4}$$

The exact form of these equations is not needed to understand the attack and can be found

in Appendix A.1. If we find a tuple (s, s', b, b') that satisfies Equations (3) and (4), we have an inner collision just before absorbing m_0 and m'_0.

The 221 equations in (4) are independent of b_i and b'_i and must hence be satisfied by the bits of s and s'. A pair $\{s, s'\}$ that satisfies these equations is a proto-collision. So we first try to find proto-collisions in the birthday set and then values of b and b' that satisfy Equations (3). To do so, we could store the birthday set in a hash table, ordering the states (and corresponding inputs) according to the 221-bit values defined by $(q'_{27}(s)||q'_{28}||\cdots||q'_{248})$. We call these the *birthday set coordinates*. Storing this hash table would cost an excessive amount of memory. However, we can use the work of Paul C. van Oorschot and Michael J. Wiener [vOW94] to find collisions without storing and sorting all computed states.

When we find a proto-collision $\{s, s'\}$, forming an inner collision still requires values of b, b' that satisfy the 26 equations of (3). As we have 26 equations and only 18 variables, this will often not be possible. This means we will have to find multiple proto-collisions to find an inner collision.

Given a proto-collision $\{s, s'\}$, the probability of satisfying remaining equations by trailing blocks b, b' is approximately 2^{-10}. The derivation of this probability can be found in Appendix A.2.

So for a given number of attemps q, the probability to have a proto-collision is approximately $\binom{q}{2}/2^{221} \approx q^2/2^{222}$ and the probability that for this proto-collision trailing blocks b and b' can be found that lead to an inner collision is 2^{-10}. It follows that the probability to get an inner collision in q attempts is close to $q^2/2^{232}$. Assuming that we can limit the computational workload per input to 1 duplex call and neglecting the computational complexity of the collision finding process, the success probability is $N^2/2^{232}$.

One may consider extending this attack by converting a proto-collision into an inner collision with sequences of two trailing blocks rather than simple trailing input blocks. However, equations corresponding to those in (3) and (4) would have algebraic degree 4 there would be many more equations involving the bits of the trailing input blocks. However, as the success probability of our attack, $N^2/2^{232}$ is only a factor 2^8 short of Claim 1, we think reducing the injection of input bits in unkeyed absorbing to 9 bits per two duplex calls is justified as a safety margin.

Adapting this attack the nominal unkeyed absorbing in Subterranean for a success probability higher than that in Claim 1 would require the construction of equations spanning 4 Subterranean rounds. We believe this to be infeasible.

3.4.3 Fixed points

A duplex-level fixed point consists of a state value s and two input block sequences A and B and has the following properties. The value s is the state after absorbing the sequence A, where A may be empty. After absorbing the non-empty sequence B starting from state s, the state must again have value s. Such a fixed point would allow generating an infinite set of input sequences $A, A||B, A||B||B, \ldots$ all colliding in the same state and would therefore generate the same output sequences.

Finding a duplex-level fixed point (s, A, B) could be done in two phases. In a first phase we can try finding a pair (s, B) and in the second phase a matching value of A. The complexity of this undertaking increases with the number of blocks in the sequence B. The simplest possible case is when B has a single block, where it comes down to finding a fixed point in two-round Subterranean, where the 9 bits of B are degrees of freedom. The expected number of such fixed points is 2^9. For B consisting of n blocks, we have to find a fixed point in $2n$-round Subterranean, with $9n$ degrees of freedom, leading to an expected 2^{9n} solutions. We expect the difficulty of finding pairs (s, B) to increase rapidly with the

number of blocks of B due to the increasing number of Subterranean rounds that must be covered.

However, the difficulty lies mostly in the second phase, where, given a set of states S, an input block sequence A must be found that would give rise to a state in S. Such an undertaking is similar to generating inner collisions as described in Section 3.4.2, but with less degrees of freedom per round for the attacker. We therefore believe duplex-level fixed points form no threat for the security of Subterranean.

3.4.4 Differential properties

For duplex call sequences of equal length one may try to generate an inner collision by exploiting a differential or trail with high DP. As an inner collision must be obtained in 248 bits of the state and the adversary can choose 9 bits per duplex call in each of the two input block sequences, it is unlikely that starting from some given state, there exist colliding input sequences of less than $248/(2 \times 9) \approx 14$ blocks. Clearly, there are $2^{9\times14} = 2^{126}$ input block sequences and just trying them all would just be a generic attack. Doing this in less calls in a systematic way would require controlling the propagation of the difference through the rounds and hence having some kind of high probability differential in Subterranean from the input blocks to the state. We believe such differentials do simply not exist.

One could imagine applying an attack that exploits differential properties combined with backtracking. This was done by Thomas Fuhr and Thomas Peyrin for mounting inner collision attacks on RadioGatún [FP09], almost breaking its security claim. Moreover, RadioGatún has a round function very similar to that of Subterranean. However, in backtracking attacks the degrees of freedom for the attacker play a very important role, and these are much higher in RadioGatún than in Subterranean. Namely, RadioGatún absorbs 3 words per round in a 19-word state (mill) of 19 words, while Subterranean absorbs 8 bits per two rounds in a 257-bit state.

3.5 State-recovery attacks

Subterranean-SAE is very similar to the CAESAR candidate Ketje Jr [BDP+14,BDP+16] that was attacked by Thomas Fuhr, Maria Naya-Plasencia and Yann Rotella in [FNR18]. This attack is a state-recovery attack on a weakened version of Ketje Jr, where the weakening consists of an increase of the rate during the wrap calls from the nominal 16 bits to 32 bits. The attack focuses on 4 consecutive rounds on Ketje Jr v1. The feasibility of the attack strongly depends on the bit positions of the outer part. In Ketje Jr the outer part covers full (5-bit) *rows* and the non-linear mapping operates at row level. This means that if in a state at the input (resp. output) of χ all bits of a row are known, one can compute the bits in that row at the output (resp. input) of χ. This fact allows an attacker to link the information between 4 consecutive rounds. In Ketje Jr v2 the definition of the outer part was changed and no longer contains full rows, greatly reducing the applicability of the attack.

In Subterranean the squeezing rate is 32 bits, so at first sight the attacks in [FNR18] may be a concern. However, two factors already make it much harder to pull of for Subterranean than for the weakened version of Ketje Jr v1:

- Subterranean has a 257-bit state and Ketje Jr only a 200-bit state.
- In Ketje Jr the χ mapping is applied on *rows* of 5 bits. In Subterranean, χ is applied on all state bits arranged in a single circle.

From the reduction in effectiveness of attacks from Ketje Jr v1 and Ketje Jr v2, we learned that it is a good idea to choose the positions of output bits *far from each other*.

Our choice of output positions does exactly that when seen in the context of state-recovery attacks. It takes the knowledge of at least two consecutive bits at the input (output) of χ to determine a single bit at its output (input) and by our choice of output bits, we avoid the propagation of knowledge of individual bit values across χ.

On top of that, we added another hurdle to frustrate state-recovery attacks: instead of simply taking individual state bits as output bits z_i, we construct each output bits as the sum of 2 state bits. This is inspired by the stream cipher Trivium [Can06], where the output bit consists of the sum of three state bits. Hence, to obtain 32 output bits, we take 64 state bits and add them in pairs. While information-technically an attacker gets the same amount of information per round as in the case of taking 32 consecutive state bits, it is much harder to exploit this information. In particular, most techniques used in [FNR18] no longer work and it appears that Subterranean has a comfortable safety margin with respect to state-recovery attacks. In particular, the algebraic expression of the output bits gets twice the number of monomials, exploding the cost of (for instance) Guess-and-Determine techniques.

3.6 Bias in the keystream

In this section we investigate possible biases in the keystream. Here, we refer to biases in linear combination of output bits such as found by Brice Minaud in AEGIS [Min14] or Tomer Ashur et al. in MORUS [AEL+18]. For Subterranean this would mean the following. We write an output stream Z as a sequence of 32-bit blocks $z_0, z_1, z_2 \ldots$ and define a mask sequence U that consists of a sequence of, 32-bit masks $u_0, u_1, u_2 \ldots u_{n-1}$ for some n. This mask sequence U defines a (binary) parity of a sequence Z as $U^{\mathrm{T}}Z = \sum_{0 \le i < n} u_i^{\mathrm{T}} z_i$. We can *shift* this mask to a later position in Z and compute a similar parity. We write $U^{\mathrm{T}}(Z \ll q) = \sum_{0 \le i < n} u_i^{\mathrm{T}} z_{q+i}$. If we have a keystream Z of m blocks, a parity determined by an n-block mask U can be computed on $m-n$ positions. If we have multiple keystreams Z of length m_i, this parity can be computed on $\sum_i (m_i - n)$ positions.

For AEGIS and MORUS, masks U were found so that $U^{\mathrm{T}}Z$ exhibits a bias, i.e., its value has higher probability to be 0 than 1 or vice versa. This is equivalent to saying that $U^{\mathrm{T}}Z$ is correlated to 0 with positive or negative correlation C. To detect a bias in an output Z with some given correlation C, one needs to compute C^{-2} parities implying at least C^{-2} output blocks. So for example to detect a bias with correlation $1/1000$ we need about a million output blocks.

For any given mask U we can form the algebraic expression of $U^{\mathrm{T}}Z$ in terms of the bits of the state value at the point of the first mask of U. If this algebraic expression is a balanced function, $U^{\mathrm{T}}Z$ has zero bias and this is the case If it contains at least one degree-1 monomial s_i that does not appear in any other monomial.

In Subterranean, every bit of z_t is the sum of two state bits of s^t. If U consists of a single block u_0, the expression of $U^{\mathrm{T}}Z$ in terms of the bits of s^0 is simply a sum of bits and hence balanced.

We will now explain the case of a two-block mask $U = u_0 u_1$. The expression for $u_0^{\mathrm{T}} z_0 + u_1^{\mathrm{T}} z_1$ is a sum of bits of s^0 and bits of s^1. One can convert the latter to bits of s^0 by substitution using the round function:

$$s_j^1 = s_i^0 + \delta_i + s_{i+3}^0 + s_{i+8}^0 + (s_{i+1}^0 + 1)s_{i+2}^0 + (s_{i+4}^0 + 1)s_{i+5}^0 + (s_{i+9}^0 + 1)s_{i+10}^0\,, \qquad (5)$$

with $i = 12 \times j$.

The 64 state bit positions used for generating the output bits z_i are the elements of the multiplicative subgroup $\langle 12^4 \rangle$ of $(\mathbb{Z}/257\mathbb{Z})^*$ of order 64 generated by $12^4 = 176$.

So the bits of s^0 corresponding to $u_0^{\mathrm{T}} z_0$ have indices i in $\langle 12^4 \rangle$. The bits of s^1 due to the term $u_0^{\mathrm{T}} z_0$ have indices j, also in $\langle 12^4 \rangle$. After converting them to bits of s^0 using (5), each bit s_j^1 leads to at least one bit $s_{12\times j}^0$, namely the leftmost at the righthand side. As

$j \in \langle 12^4 \rangle$, this bit of s^0 has an index i in $12\langle 12^4 \rangle$. It follows that the degree-1 monomials due to $u_0^{\mathrm{T}} z_0$ and those due to $u_1^{\mathrm{T}} z_1$ have non-overlapping indices and therefore there is always at least one degree-1 monomial that does not appear in any other monomial. In other words, there is no bias for masks Z of less than 3 blocks.

For 3-block and 4-block masks, the reasoning above generalizes in that $u_2^{\mathrm{T}} z_2$ generates degree-1 monomials with indices in $12^2\langle 12^4 \rangle$ and $u_3^{\mathrm{T}} z_3$ degree-1 monomials with indices in $12^3\langle 12^4 \rangle$. This provides evidence that there is probably no bias for masks Z of less than 5 blocks and we believe there is is no bias in Z measurable from output sequences of 2^{96} blocks or less.

3.7 Differential trail analysis

We investigated the differential propagation in Subterranean in the form of differential trails. For this, we used an approach inspired by the work of Silvia Mella et al. in [MDA17] that was later also applied to Xoodoo in [DHVAVK18]. It consists in generating all *trail cores* over a small number of rounds up to some *weight*. A trail core is trail where the initial difference and the final difference are left unspecified and its weight is the minimum of the weight of all trails compatible with that trail core. The weight w of a trail Q corresponds to its differential probability $\mathrm{DP}(Q)$ and is given by $w(Q) = \log_2 \mathrm{DP}(Q)$, so $\mathrm{DP}(Q) = 2^{-n}$ implies $w(Q) = n$.

One-round trails correspond with differentials and even for relatively small weights there are huge numbers of trails. This is due to the fact that they can be reduced to differentials over the non-linear layer χ only. For two-round trail cores, the numbers decrease as the diffusion of the linear mapping $\lambda = \pi \circ \theta$ in between the χ layers kicks in. However, due to the limited mixing in θ, the histogram is still quite dense. Starting from 3 rounds, the landscape of trail cores becomes quite sparse due to the interaction of the linear step λ of consecutive rounds. This can be seen in the 3-round trail core weight histogram in Table 1.

Table 1: Trail core weight histogram for 3 rounds of Subterranean. We report only on the number of trail cores that are different modulo translation: each unit in this table corresponds to 257 trail cores.

weight	25	28	29	30	32	33	34	35	36	37	38	39
# trail cores	1	1	2	3	2	1	5	6	4	9	12	17

We report on the bounds for 1 up to 8 rounds in Table 2. For 1 up to 3 rounds the bounds are tight as we have trails with the reported weight. For 4 rounds we know there are no trails with weight less than 49 and have found a trail with weight 58. For more than 4 rounds, no trails were found but the space was scanned up to the given limits. The 3-round trail core with weight 25 and the 4-round trail core with weight 58 can be found in Appendix B.

Table 2: Lower bounds for the weight of differential trails in Subterranean for different number of rounds.

# rounds:	1	2	3	4	5	6	7	8
lower bound:	2	8	25	$[49, 58]$	≥ 54	≥ 65	≥ 70	≥ 98

Based on our understanding of the propagation through χ and λ, we expect similar bounds for linear trails. We leave their analysis as future work.

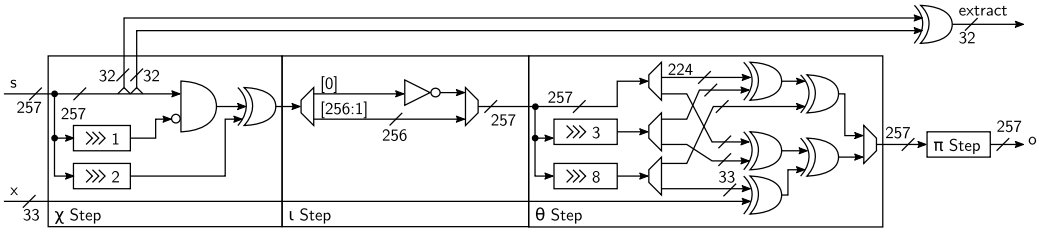

Figure 3: Subterranean duplex combinatorial circuit.

3.8 Time-Memory-Data Trade-offs

When squeezing an output, Subterranean behaves like a stream cipher and hence it may be subject to Time-Memory trade-off attacks as specified by Steve Babbage in [Bab95]. Those attacks can recover the internal state given resources $M = T = \sqrt{N}$ with M the memory complexity and T the time complexity and N the stream cipher state space. As for Subterranean $N = 2^{257}$, these attacks are not a threat for our claimed security strength of 128 bits. In 2000, Alex Biryukov and Adi Shamir [BS00] improved the trade-off by bringing the data complexity D and computation in the pre-processing phase P into the equation. The invariances of their trade-off become $TP = N$ and $MD = N$. As we limit the data complexity to $D < 2^{96}$ and have $N = 2^{257}$, this still does not jeopardize the claimed security strength of 128 bits.

4 Implementation

Since Subterranean 2.0 is designed for hardware, we discuss a hardware architecture aimed at ASICs, report on its resource usage and discuss why we believe it is suited for low-energy implementations. We also illustrate how to implement Subterranean relatively efficient in software using the π *procrastination* technique. The code for the hardware implementation can be found in the NIST Lightweight competition website `https://cs.ru.nl/~joan/subterranean.html` together with a software reference code, mainly used to debug the hardware implementation.

4.1 Hardware architecture

We design as a proof of concept an hardware circuit that can perform duplex, extract an output block and perform block-level absorbing, encryption and decryption. Also, we compare with some other lightweight duplex-based designs.

Figure 3 illustrates the Subterranean duplex combinatorial circuit that basically just extends the one in Figure 1, with the extraction logic and the injection of x in the θ step. The latter allows extending some of the 3-input XORs of θ step to 4-input XOR. These 4-input XOR gates can be synthesized directly to 4-input XOR cell, or more commonly, to three 2-input XOR cells. When done with 2-input XOR gates, as one can see in Figure 3, the addition of the input injection does not affect the total gate delay of the round logic.

We chose to add the extraction logic in this abstraction layer to have a single combinatorial block performing all cipher processing at duplex level, minus the padding.

The Subterranean duplex circuit in Figure 4 builds on top of the duplex combinatorial circuit by adding the 257-bit state register and the operation logic handling encryption, decryption and padding. The state register can be loaded with all zeroes, or by the round function applied to its previous value and the input block. The circuit computes the value of the injected string x by padding a string σ formed from the first bytes of din. The length of σ in bytes shall be indicated at dinSize. In case of the decrypt operation the

circuit XORs the first bytes of din to the first bytes of ex to form σ before padding. The output dout contains ex in case of (un)keyed absorbing and the XOR of din and ex in case of encrypt and decrypt.

Figure 4 shows the subterranean duplex hardware architecture with single-round combinatorial logic. This architecture can be expanded by unrolling 2 or even 4 rounds in the combinatorial logic, with per round a clone of the circuit of Figure 3 and additional control logic indicating how many rounds must be executed. This unrolling allows increasing the throughput and decreasing the power consumption per encrypted/decrypted bit at the cost of increased area and latency.

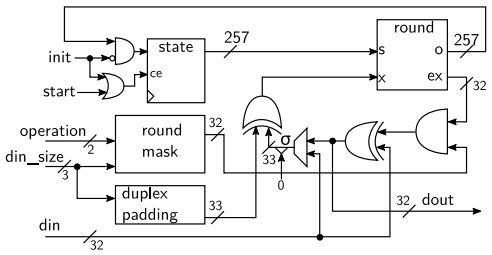

Figure 4: Subterranean single-round duplex logic.

Unit	Area (GE)	Area (%)
Subterranean duplex	5160	100
Round	2040	39.5
Registers	1463	28.4
State MUX	943	18.3
din padding	16	0.3
dout mask	9	0.2
Duplex din MUX	75	1.5
dout encrypt/decrypt	96	1.9
Buffer	518	9.9

Table 3: NanGate OpenCell 45nm typical area costs for Subterranean single-round duplex logic without flattening.

4.2 Hardware results

We synthesized the Subterranean circuit of Figure 4 and the 2- and 4-round unrolled variants, plus some similar authenticated encryption schemes in the ASIC cells NanGate OpenCell 45nm typical through the open source tool Yosys version 0.9 [Wol] and also, thanks to Silvia Mella, STMicroelectronics 40 nm technology with Synopsys Design Compiler and PrimePower tools. During our synthesis we opted for a clock period of 10 ns, which makes the tool optimized for area, but not for critical path.

We first synthesized with NanGate OpenCell 45nm typical the architecture in Figure 4 with flattening disabled in Table 3. The flattening option in hardware tools takes the entire architecture code of all units and merges into one code. This technique enables the tool to optimize across the boundaries of all units, thus optimizing the entire circuit. Since we want to know the cost of each part in the Subterranean duplex circuit, we disabled flattening only for Table 3.

Table 3 reveals that the largest building block is the duplex combinatorial logic. The $257 \times 3 = 771$ XOR gates in θ and χ at 2 GE (gate equivalent) per XOR already give 1542 GE. Input injection and output extraction add 64 more XOR gates, adding 128 more GE. The second largest building block is the state register taking in total 257 flip-flops (ff) each accounting for 5.67 GE. Next is the multiplexer that chooses between keeping the state register value, updating through the round function or loading with zeroes. Yosys has a number of limitations without which one could reduce the area somewhat. First, it would be possible to eliminate the multiplexer by replacing the flip-flops in the registers with scan-ff as they have an embedded multiplexer functionality. Second, Yosys currently does not support cells with 2 outputs. Having such cells would allow removing the inverters used in χ circuit, since flip-flops have regular and negated outputs.

For comparison and evaluation, in Table 5, we used the results from STMicroelectronics 40 nm technology with flattening optimization. In case of Subterranean we report on single-round, 2- and 4-round unrolled duplex logic. The similar schemes we compare with

in Table 5 are Gimli [BKL+17], with 3 different unrolling variants and Ketje Jr and Ketje Sr [BDP+16]. Note however that for these schemes we only report on the area of the round logic and the state register. Even though they have similar lightweight goals, they have different characteristics that are explained and illustrated in Table 4. Gimli has 4 different rounds favoring a 4-round unrolled round logic but requires additional multiplexers in single-round and 2-round unrolled round logic.

Table 5 reveals that the single-round Subterranean duplex has area very close to the smallest scheme, Ketje Jr, but has more than twice its throughput. Therefore Subterranean is a very good option for environments that require lightweight crypto in hardware with high throughput requirements.

Table 4: Characteristics of some duplex-based AE schemes. Cost for message and session denote the fixed cost per message and session respectively, assuming 128-bit tags.

Scheme	b (bits)	block (bits)	computational cost in # rounds per block	message	session
Subterranean-SAE	257	32	1	12	17
Gimli AEAD	384	128	24	24	n.a.
Ketje Jr	200	16	1	12	13
Ketje Sr	400	32	1	12	9

Table 5: Synthesis results for STMicroelectronics 40 nm for frequency 100 MHz

Implementation	Area (GE)	Power (μW) Average	Peak	Energy (picoJoule), per cycle	bit	message	session
Subterranean–1	4165.5	432.4	777	4.324	0.135	51.89	73.51
Subterranean–2	6680.6	503.1	1074	5.031	0.079	30.18	45.28
Subterranean–4	12163.1	654.1	1998	6.541	0.051	19.62	32.70
Gimli–1	5242.6	720.0	874	7.200	1.350	172.80	n.a.
Gimli–2	6949.8	813.9	1078	8.139	0.760	97.67	n.a.
Gimli–4	10209.3	935.1	1468	9.351	0.438	56.10	n.a.
Ketje Jr	3186.5	362.5	521	3.625	0.226	43.50	47.13
Ketje Sr	6278.1	721.1	1068	7.211	0.225	86.53	64.90

4.3 On the low energy aspect of Subterranean

As can be seen in Table 5, the energy consumption per bit (of plaintext, ciphertext, AD, key or nonce) of the Subterranean implementations is quite competitive. This is even the case when compared to the Ketje schemes that were specifically designed for the same purpose: Subterranean–2 is about 3 times as energy-efficient than Ketje Sr while they both are designed or 128-bit security strength and consume similar area. Moreover, Subterranean–2 requires less energy for setting up a session and has less fixed overhead per message. One can get insight in the energy consumption numbers of Table 5 by associating with an architecture a 2-dimensional metric we introduce here: the *bit switch cost*. The processing of an input block takes a number of cycles and we expect the energy consumption of a cycle to be dominated by the update of the state register and the consumption of the combinatorial round logic. The bit switch cost has two components:

- The number of state register flip-flop updates per data block, divided by the size of the data block. This is expressed in flip-flop updates (FF).
- The number of round function executions per data block, divided by the size of the data block. This is expressed in so-called round slices (RS).

The energy consumption per FF depends on the technology but is independent of the cryptographic function and architecture. The energy consumption per RS depends on the technology, the details of the round function such as the amount of gates and the circuit depth and the hardware architecture: amount of unrolling, the presence of multiplexers or the possible application of pipelining.

The bit switch cost is useful for comparing different architectures of the same cryptographic function, but also allows comparing those of different cryptographic functions, at least if they have comparable round functions.

Subterranean–1, with its 257-bit state and round and 32-bit data input/output per round has $257/32 \approx 8$ FF and similarly 8 RS. Unrolling does not impact the RS score but does divide the RS by the number of unrolled rounds, so Subterranean–2 has 4 FF and 8 RS and Subterranean–4 has 2 FF and 8 RS. A naive interpretation of the energy numbers of Table 5 would be that an FF consumes about 0.014 pJ and an RS 0.003 pJ. For Ketje, with its 200/400-bit state and round and 16/32-bit data per round the score has $200/16 \approx 12.5$ FF and similarly 12.5 RS. Filling in the energy consumption numbers for FF and RS obtained from Subterranean would give an energy consumption per bit of $12.5 \times (0.014 + 0,003) = 0.2125$. This is very close to the value of the simulation in Table 5. That makes sense because the round functions of Ketje and Subterranean are very similar.

For Gimli–1, with its 384-bit state and round and 128-bit data per 24 rounds the bit switch cost gives $384/128 \times 24 = 72$ FF and similarly 72 RS. In the unrolled version this is reduced to 36 FF and 72 RS for Gimli–2 and 18 FF and 72 RS for Gimli. Taking the FF energy consumption interpolated from Subterranean, an RS would cost 0.0047 in Gimli–1, 0.0036 in Gimli–2 and 0.0026 in Gimli–4. That makes sense because Gimli has different round functions for different round indices necessitating the insertion of multiplexers in Gimli–1 and somewhat less in Gimli–2. We do not have hardware numbers of two other very similar schemes in the NIST lightweight competition Xoodyak, Ascon, but their bit switch cost gives an idea of what to expect. In single-round architectures, Xoodyak has $384/192 \times 12 \approx 24$ FF and 24 RS, Ascon has $320/64 \times 6 = 30$ FF and 30 RS. The FF component can be reduced by round unrolling. A RS in Xoodyak has the same number of gates as Subterranean but a slightly higher complexity and an RS of Ascon has slightly more gates, so we expect their energy consumption per bit to be considerably higher than Subterranean, for same number of unrolled rounds.

Single-round architectures of block cipher based schemes tend to have much higher bit switch cost than Subterranean. A mode that can process 1 n-bit block per block cipher call with n the block cipher block length has bit switch cost r FF and r RS. Most modes require 2 block cipher calls per n-bit block and there the bit switch cost increases to $2r$ FF and $2r$ RS. Lightweight block ciphers tend to have light rounds and so an RS contributes less, but have very many rounds. Round unrolling can reduce the number of FFs but will not decrease the number of RS and will typically even increase its energy consumption from some point on due to larger circuit depth.

4.4 Speeding up software by procrastination of π

The techniques in this section were explored and refined by Thomas van der Burgt in his bachelor thesis at the Radboud University [vdB19].

Steps χ, θ and ι lend themselves well for software implementations. The bit permutation π on the other hand requires the manipulation of individual bits and is costly on all CPUs.

Using the translation-invariance properties discussed in Section 3.2.1, we provide an alternative description where variants of χ and θ operate on a dynamically changing state and without π. Basically, we push π in front of us, hence the name π *procrastination*.

Let γ be shortcut notation for $\theta \circ \iota \circ \chi$ and $\gamma^{(t)}$ for $\pi^{-t} \circ \gamma \circ \pi^t$, then applying Lemma 7

we have for all $n \in \mathbb{N}$,

$$\mathrm{R}^n = \pi^n \circ \gamma^{(n-1)} \circ \gamma^{(n-2)} \circ \dots \gamma^{(1)} \circ \gamma^{(0)} \ . \tag{6}$$

Programming $\gamma^{(t)}$ instead of the round function allows procrastinating π eternally. We can again split up $\gamma^{(t)}$ in its components:

$$\gamma^{(t)} = \pi^{-t} \circ \theta \circ \pi^t \circ \pi^{-t} \circ \iota \circ \pi^t \circ \pi^{-t} \circ \chi \circ \pi^t = \theta^{(t)} \circ \iota^{(t)} \circ \chi^{(t)} \ ,$$

with $\theta^{(t)}$ and $\chi^{(t)}$ defined as in Lemma 5:

$$s_i \leftarrow s_i + (s_{i+12^t} + 1) s_{i+2\times 12^t} \quad \text{and} \quad s_i \leftarrow s_i + s_{i+3\times 12^t} + s_{i+8\times 12^t 8} \ .$$

Both $\chi^{(t)}$ and $\theta^{(t)}$ can be coded efficiently with bitwise Boolean and shift instructions. $\iota^{(t)}$ is the same as ι as π does not move bit 0 and requires a single bitwise XOR instruction.

For input injection and output extraction, we need to locate the corresponding bit positions in a register R containing the evolving state s^t. The choice of $\langle 12^4 \rangle$ for these positions allows doing this reasonably efficiently.

Using Equation (6), we have that for all $0 \leq i \leq 256$ and any $t \in \mathbb{N}$,

$$s_i^t = \mathsf{R}[12^t i] \, .$$

Bit positions in the register evolve by the multiplication by 12, that generates $(\mathbb{Z}/257\mathbb{Z})^*$.

At round 0, the indices are $\langle 12^4 \rangle = [1, 176, 136, \dots, 92]$. At round 1, they are the coset of $\langle 12^4 \rangle$ multiplied by 12: $[12, 56, 90, \dots, 76]$. In general, at round t, they are the coset of $\langle 12^4 \rangle$ multiplied by $12t$. Clearly $12^4 \langle 12^4 \rangle = \langle 12^4 \rangle$ and hence the set of indices cycles every 4 rounds.

In software we can hardcode them as four 64-byte arrays and take care of the initial offset by a pointer to the first position that increments by one position each 4 rounds.

5 Conclusions

Subterranean is a cryptographic primitive that can be used for all symmetric cryptographic operations including stream encryption, MAC computation, key derivation, hashing and duplex-based authenticated encryption.

Subterranean has a modular design with at its core is a duplex object with a relatively small state and a round function with a very simple description optimized for dedicated hardware. The round function has some nice features such as a dense inverse and algebraic degree 2.

Duplex has the advantage that it requires no fixed key as block ciphers do and hence offers better leakage resilience features. The SAE mode supports session-based authenticated encryption, or seen from a different perspective, supports intermediate tags, to limit the amount of buffering needed at unwrapping end required when not releasing unverified deciphered ciphertext.

6 Acknowledgements

We thank Silvia Mella for all the ASIC synthesis, simulation, testing and results for the STMicroelectronics 40 nm technology.

We thank Gilles Van Assche, Michaël Peeters, Seth Hoffert, Ronny Van Keer for inspiring the division in a duplex object and modes on top of it, Bart Mennink for his advice on the provable security aspects.

We thank Thomas van der Burgt for detecting errors in early specifications and reference code and for his contribution to the π procrastination technique.

We thank Xilinx for the Vivado and ISE software licenses and donating the ZedBoard. They were useful to obtain the FPGA results and evaluating the hardware design. We thank Lejla Batina for letting us use her side-channel lab and for the infrastructure supplied during the ZedBoard tests.

Joan Daemen, Alireza Mehrdad and Yann Rotella (when working at Radboud University) are supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA.

Pedro Maat Costa Massolino is supported by the Technology Foundation STW (project 13499 - TYPHOON), from the Dutch government.

References

[AEL+18] Tomer Ashur, Maria Eichlseder, Martin M. Lauridsen, Gaëtan Leurent, Brice Minaud, Yann Rotella, Yu Sasaki, and Benoît Viguier. Cryptanalysis of MORUS. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 35–64. Springer, 2018.

[Bab95] Steve Babbage. Improved "exhaustive search" attacks on stream ciphers. In *European Convention on Security and Detection*, pages 161–166. IEEE Conference Publication, 1995.

[BCC11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of keccak and *Luffa*. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2011.

[BDH+17] G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer. Farfalle: parallel permutation-based cryptography. *IACR Trans. Symmetric Cryptol.*, 2017(4):1–38, 2017.

[BDP+14] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. CAESAR submission: Ketje v1.1, March 2014. https://keccak.team/ketje.html.

[BDP+16] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. CAESAR submission: Ketje v2, September 2016. https://keccak.team/ketje.html.

[BDPA06] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. RadioGatún, a belt-and-mill hash function. Second Cryptographic Hash Workshop, Santa Barbara, August 2006. http://radiogatun.noekeon.org/.

[BDPA07] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. Ecrypt Hash Workshop 2007, May 2007. also available as public comment to NIST from http://www.csrc.nist.gov/pki/HashWorkshop/Public_Comments/2007_May.html.

[BDPA11a] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Cryptographic sponge functions, January 2011. https://keccak.team/files/SpongeFunctions.pdf.

[BDPA11b] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak reference, January 2011. http://keccak.noekeon.org/.

[BDPA14] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The making of KECCAK. *Cryptologia*, 38(1):26–60, 2014.

[Ber08] D. J. Bernstein. Chacha, a variant of Salsa20. Workshop Record of SASC 2008, 2008.

[BKL+17] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli : A cross-platform permutation. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 299–320. Springer International Publishing, 2017.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.

[Can06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier López, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[CDGP93] Luc J. M. Claesen, Joan Daemen, Mark Genoe, and G. Peeters. Subterranean: A 600 mbit/sec cryptographic VLSI chip. In *Proceedings 1993 International Conference on Computer Design: VLSI in Computers & Processors, ICCD '93, Cambridge, MA, USA, October 3-6, 1993*, pages 610–613. IEEE Computer Society, 1993.

[Dae95] J. Daemen. *Cipher and hash function design strategies based on linear and differential cryptanalysis, PhD Thesis*. K.U.Leuven, 1995.

[DC98] Joan Daemen and Craig S. K. Clapp. Fast hashing and stream encryption with PANAMA. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 1998.

[DGV91] Joan Daemen, René Govaerts, and Joos Vandewalle. A framework for the design of one-way hash functions including cryptanalysis of damgård's one-way function based on a cellular automaton. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings*, volume 739 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 1991.

[DHAK18] J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer. Xoodoo cookbook. *IACR Cryptology ePrint Archive*, 2018:767, 2018.

[DHVAVK18] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. The design of xoodoo and xoofff. *IACR Transactions on Symmetric Cryptology*, 2018(4):1–38, Dec. 2018.

[DR02] J. Daemen and V. Rijmen. *The design of Rijndael — AES, The Advanced Encryption Standard*. Springer-Verlag, 2002.

[FLM10] Niels Ferguson, Stefan Lucks, and Kerry A. McKay. Symmetric states and their structure: Improved analysis of cubehash. Cryptology ePrint Archive, Report 2010/273, 2010. https://eprint.iacr.org/2010/273.

[FNR18] Thomas Fuhr, María Naya-Plasencia, and Yann Rotella. State-recovery attacks on modified ketje jr. *IACR Trans. Symmetric Cryptol.*, 2018(1):29–56, 2018.

[FP09] Thomas Fuhr and Thomas Peyrin. Cryptanalysis of radiogatún. In Orr Dunkelman, editor, *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 2009.

[LIM19] Fukang Liu, Takanori Isobe, and Willi Meier. Cryptanalysis of subterranean-sae. *IACR Cryptology ePrint Archive*, 2019:879, 2019.

[MDA17] Silvia Mella, Joan Daemen, and Gilles Van Assche. New techniques for trail bounds and application to differential trails in keccak. *IACR Trans. Symmetric Cryptol.*, 2017(1):329–357, 2017.

[Men18] Bart Mennink. Key prediction security of keyed sponges. *IACR Trans. Symmetric Cryptol.*, 2018(4):128–149, 2018.

[Min14] Brice Minaud. Linear biases in AEGIS keystream. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2014.

[MRH04] U. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In M. Naor, editor, *Theory of Cryptography - TCC 2004*, number 2951 in Lecture Notes in Computer Science, pages 21–39. Springer-Verlag, 2004.

[RSS11] T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In *Eurocrypt 2011*, pages 487–506, 2011.

[TS14] M. M. I. Taha and P. Schaumont. Side-channel countermeasure for SHA-3 at almost-zero area overhead. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2014*, pages 93–96. IEEE Computer Society, 2014.

[vdB19] T. van der Burgt. *Implementing the NIST lightweight cryptography candidates Xoodyak and Subterranean 2.0, Bachelor Thesis*. Radboud University, 2019.

[vOW94] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, and Ravi S. Sandhu, editors, *CCS '94, Proceedings of the 2nd ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 2-4, 1994*, pages 210–218. ACM, 1994.

[Wol] Clifford Wolf. Yosys Open SYnthesis Suite. http://www.clifford.at/yosys/.

A Details of dedicated inner collision cryptanalysis

In this appendix, we provide necessary material to verify the attack depicted in Section 3.4.2

A.1 System of equations

Here we give equations where bits b_2, b'_2, b_5 and b'_5 intervene. Those bits are respectively injected at position 136 and 134. This yields equations of the form:

$$\begin{cases} q_{124}(s)+q_{124}(s') &= b_5 s_{133} + b'_5 s'_{133} \\ q_{125}(s)+q_{125}(s') &= b_5 s_{135} + b'_5 s'_{135} \\ q_{126}(s)+q_{126}(s') &= b_5 + b'_5 + b_2 s_{135} + b'_2 s'_{135} \\ q_{127}(s)+q_{127}(s') &= b_2 s_{137} + b'_2 s'_{137} \\ q_{128}(s)+q_{128}(s') &= b_2 + b'_2 \\ q_{129}(s)+q_{129}(s') &= b_5 s_{133} + b'_5 s'_{133} \\ q_{130}(s)+q_{130}(s') &= b_5 s_{135} + b'_5 s'_{135} \\ q_{131}(s)+q_{131}(s') &= b_5 + b'_5 + b_2 s_{135} + b'_2 s'_{135} \\ q_{132}(s)+q_{132}(s') &= b_5 s_{133} + b'_5 s'_{133} + b_2 s_{137} + b'_2 s'_{137} \\ q_{133}(s)+q_{133}(s') &= b_5 s_{135} + b'_5 s'_{135} + b_2 + b'_2 \\ q_{134}(s)+q_{134}(s') &= b_5 + b'_5 + b_2 s_{135} + b'_2 s'_{135} \\ q_{135}(s)+q_{135}(s') &= b_2 s_{137} + b'_2 s'_{137} \\ q_{136}(s)+q_{136}(s') &= b_2 + b'_2 \end{cases}$$

By doing a Gaussian elimination, we can say that this system of equations is equivalent to

$$\begin{cases} q'_{124}(s)+q'_{124}(s') &= 0 \\ q'_{125}(s)+q'_{125}(s') &= 0 \\ q'_{126}(s)+q'_{126}(s') &= 0 \\ q'_{127}(s)+q'_{127}(s') &= 0 \\ q'_{128}(s)+q'_{128}(s') &= 0 \\ q'_{129}(s)+q'_{129}(s') &= 0 \\ q'_{130}(s)+q'_{130}(s') &= 0 \\ q'_{131}(s)+q'_{131}(s') &= 0 \\ q'_{132}(s)+q'_{132}(s') &= b_5 s_{133} + b'_5 s'_{133} \\ q'_{133}(s)+q'_{133}(s') &= b_5 s_{135} + b'_5 s'_{135} \\ q_{134}(s)+q_{134}(s') &= b_5 + b'_5 + b_2 s_{135} + b'_2 s'_{135} \\ q_{135}(s)+q_{135}(s') &= b_2 s_{137} + b'_2 s'_{137} \\ q_{136}(s)+q_{136}(s') &= b_2 + b'_2 \end{cases}$$

where $q'_{133} = q_{133} + q_{136}$, $q'_{132} = q_{135}$, $q'_{131} = q_{131} + q_{134}$, $q'_{130} = q_{130} + q'_{133}$, $q'_{129} = q_{129} + q'_{132}$, $q'_{128} = q_{128} + q_{136}$, $q'_{127} = q_{127} + q_{135}$, $q'_{126} = q_{126} + q_{134}$, $q'_{125} = q_{125} + q'_{133}$ and $q'_{124} = q_{124} + q'_{132}$.

Except for this specific behaviour of bits b_2 and b_5 that are injected at positions 136 and 134, the injected bits yield a system of 3×7 equations, concatenated with equations

exclusively in bits of s and s'. The system of equations we obtain (after applying the Gaussian elimination and reordering the equations) has the following form

$$\left\{\begin{array}{rcl}
q'_1(s) + q'_1(s') &=& b_0 + b'_0 \\
q'_2(s) + q'_2(s') &=& b_0 s_0 + b'_0 s'_0 \\
q'_3(s) + q'_3(s') &=& b_0 s_2 + b'_0 s'_2 \\
q'_4(s) + q'_4(s') &=& b_1 + b'_1 \\
q'_5(s) + q'_5(s') &=& b_1 s_{175} + b'_1 s'_{175} \\
q'_6(s) + q'_6(s') &=& b_1 s_{177} + b'_1 s'_{177} \\
&\cdots& \\
q'_{19}(s) + q'_{19}(s') &=& b_7 + b'_7 \\
q'_{20}(s) + q'_{20}(s') &=& b_7 s_{63} + b'_7 s'_{63} \\
q'_{21}(s) + q'_{21}(s') &=& b_7 s_{65} + b'_7 s'_{65} \\
q'_{22}(s) + q'_{22}(s') &=& b_5 s_{133} + b'_5 s'_{133} \\
q'_{23}(s) + q'_{23}(s') &=& b_5 s_{135} + b'_5 s'_{135} \\
q_{24}(s) + q_{24}(s') &=& b_5 + b'_5 + b_2 s_{135} + b'_2 s'_{135} \\
q_{25}(s) + q_{25}(s') &=& b_2 s_{137} + b'_2 s'_{137} \\
q_{26}(s) + q_{26}(s') &=& b_2 + b'_2 \\
q'_{27}(s) &=& q'_{27}(s') \\
q'_{28}(s) &=& q'_{28}(s') \\
&\cdots& \\
q'_{248}(s) &=& q'_{248}(s')
\end{array}\right.$$

where q'_j, for all $0 \leq j \leq 27$, are quadratic functions exclusively in bits of s and s'. The q'_j functions differs from q_j functions as we did a Gaussian elimination.

A.2 Probability of satisfying remaining equations

For any i with i different from 2 and 5, bits b_i and b'_i for different i occur in non-overlapping equations and there are 3 equations per couple (b_i, b'_i). An exception to this are equations in bits b_2, b'_2, b_5 and b'_5 that we will address afterwards. We will now work out the case for b_0 and b'_0 that corresponds to the first 3 equations.

- If $s_0 = s'_0$ and $s_2 = s'_2$, then only the difference $b_0 + b'_0$ matters. This difference is uniquely determined by the first equation and this equation can be satisfied by choosing $b_0 + b'_0$. The next two equations are then both satisfied with probability 2^{-2}.
- If $s_0 = s'_0 + 1$ or $s_2 = s'_2 + 1$, then the difference $b_0 + b'_0$ matters for the first equation, but the absolute value also matters. This uniquely determines the value of b_0 and b'_0 by using only 2 equations. The last equation is then satisfied with probability 2^{-1}.

Hence, the first three equations can be satisfied with probability

$$\frac{1}{4} \times \frac{1}{4} + \frac{3}{4} \times \frac{1}{2} = \frac{7}{16}\,.$$

This probability is the same for equations where b_1, b_3, b_4, b_6, b_7 and b_8 intervene.

For the five equations where b_2, b'_2, b_5 and b'_5 intervene, the previous analysis does not hold. To obtain the probability that these equations can be satisfied, we can look into the following 8 different events:

- $s_{137} = s'_{137}$, $s_{135} = s'_{135}$, $s_{133} = s'_{133}$;
- $s_{137} \neq s'_{137}$, $s_{135} = s'_{135}$, $s_{133} = s'_{133}$;
- $s_{137} = s'_{137}$, $s_{135} \neq s'_{135}$, $s_{133} = s'_{133}$;

- $s_{137} = s'_{137}$, $s_{135} = s'_{135}$, $s_{133} \neq s'_{133}$;
- $s_{137} \neq s'_{137}$, $s_{135} \neq s'_{135}$, $s_{133} = s'_{133}$;
- $s_{137} \neq s'_{137}$, $s_{135} = s'_{135}$, $s_{133} \neq s'_{133}$;
- $s_{137} = s'_{137}$, $s_{135} \neq s'_{135}$, $s_{133} \neq s'_{133}$;
- $s_{137} \neq s'_{137}$, $s_{135} \neq s'_{135}$, $s_{133} \neq s'_{133}$.

By using the same arguments as before, the probability of satisfying the five equations can be expressed as

$$\frac{1}{8}\left(\frac{1}{8}+\frac{1}{4}+\frac{3}{8}+\frac{1}{4}+\frac{1}{2}+\frac{1}{2}+\frac{3}{8}+\frac{1}{2}\right)=\frac{23}{64}\,.$$

Putting everything together, the probability of satisfying all remaining equations by choosing freely b_i and b'_i for i from 0 to 8 is

$$p' = \left(\frac{7}{16}\right)^7 \left(\frac{23}{64}\right) \approx 2^{-10}\,.$$

B Differential trail core examples

In this appendix we give an example of a differential trail core over 3 rounds and one over 4 rounds. The 3-round trail core is in Table 6 and it is the single one (modulo translation) that has the lowest weight, namely 25. The 4-round trail core is in Table 7. It has weight 58 and it may be that there are 4-round trail cores with weight in the interval $[49, 57]$, since there is no 4-round trail with weight 48 or less.

In Table 6 and 7, we specify the trail cores with the intermediate difference b_i before χ by the positions of their active bits and we specify the weight of the corresponding differentials over χ. This is possible because χ has algebraic degree 2 and the weight of a differential over a quadratic mapping is fully determined by its input difference. For the first round differential a trail core only specifies its output difference a_1 and the weight reported is the minimum weight over all possible differentials over χ of the form (b_0, a_1).

Table 6: 3-round differential trail core with lowest weight.

state	weight	active bit positions
a_1	2	$\{0\}$
b_1	6	$\{0, 64, 85\}$
b_2	17	$\{0, 64, 85, 91, 155, 157, 176, 221, 242\}$

Table 7: 4-round differential trail with lowest weight we found.

state	weight	active bit positions
a_1	12	$\{0, 5, 8, 10, 12, 15, 16, 18, 21\}$
b_1	7	$\{65, 66, 85, 86, 87\}$
b_2	11	$\{7, 28, 134, 198, 200, 219\}$
b_3	28	$\{16, 18, 22, 39, 54, 86, 88, 107, 118, 139, 152, 173, 188, 211, 252\}$

C Figures for Subterranean absorb, squeeze and blank

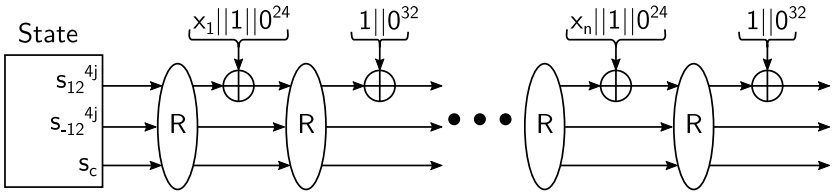

Figure 5: Subterranean absorb with operation unkeyed.

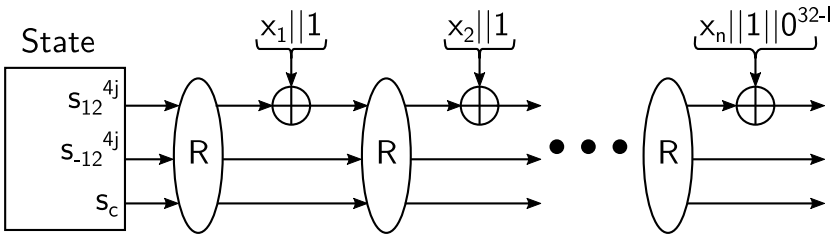

Figure 6: Subterranean absorb with operation keyed.

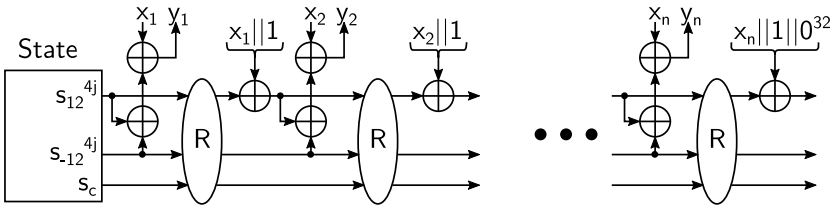

Figure 7: Subterranean absorb with operation encryption.

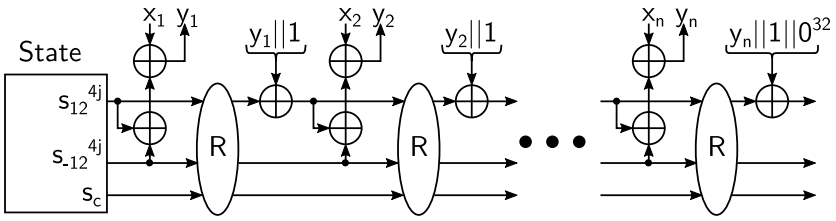

Figure 8: Subterranean absorb with operation decryption.

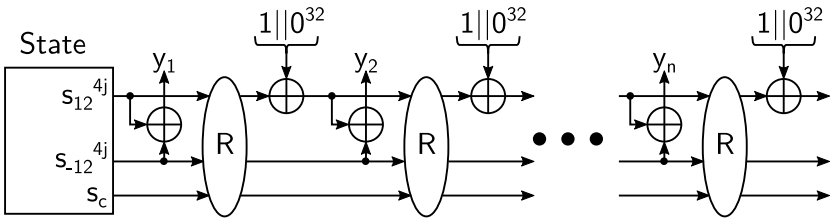

Figure 9: Subterranean squeeze.

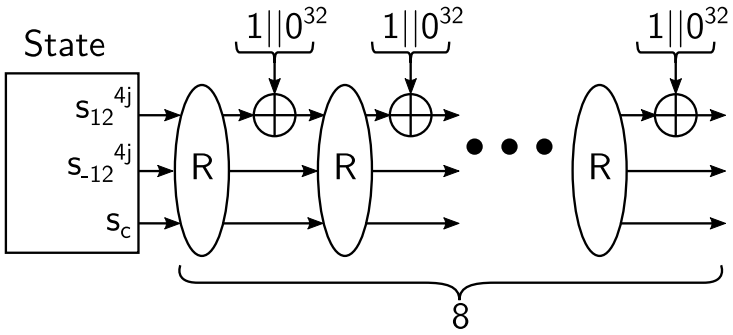

Figure 10: Subterranean blank.

D Figures for Subterranean XOF, deck and SAE

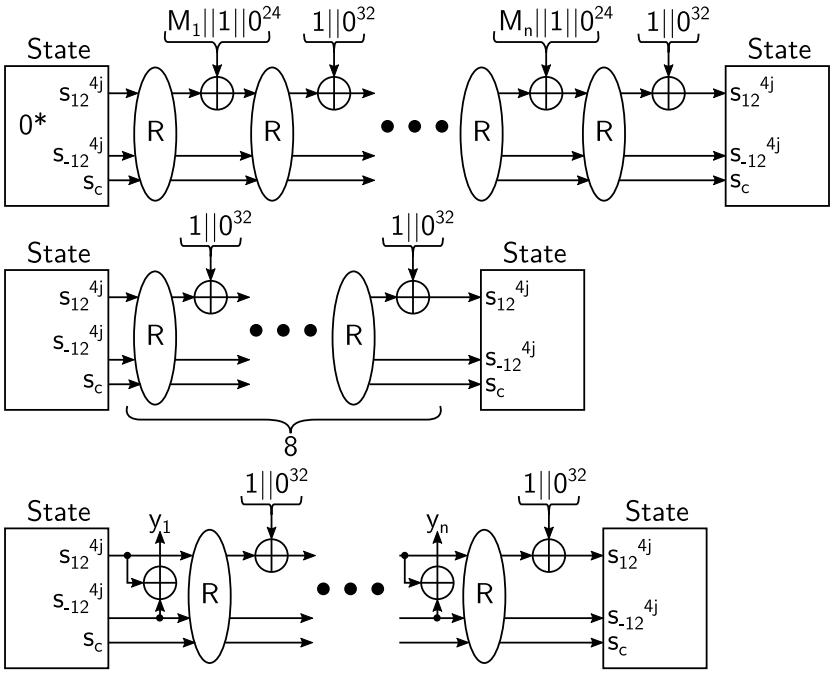

Figure 11: Subterranean XOF.

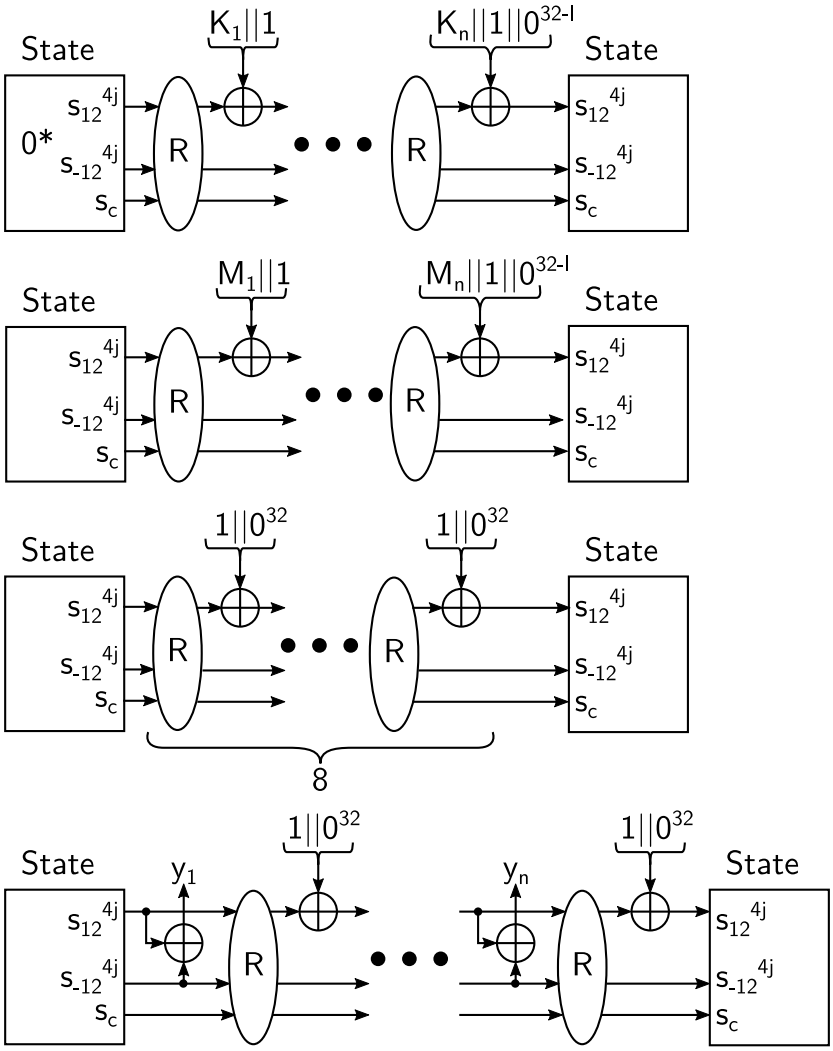

Figure 12: Subterranean deck.

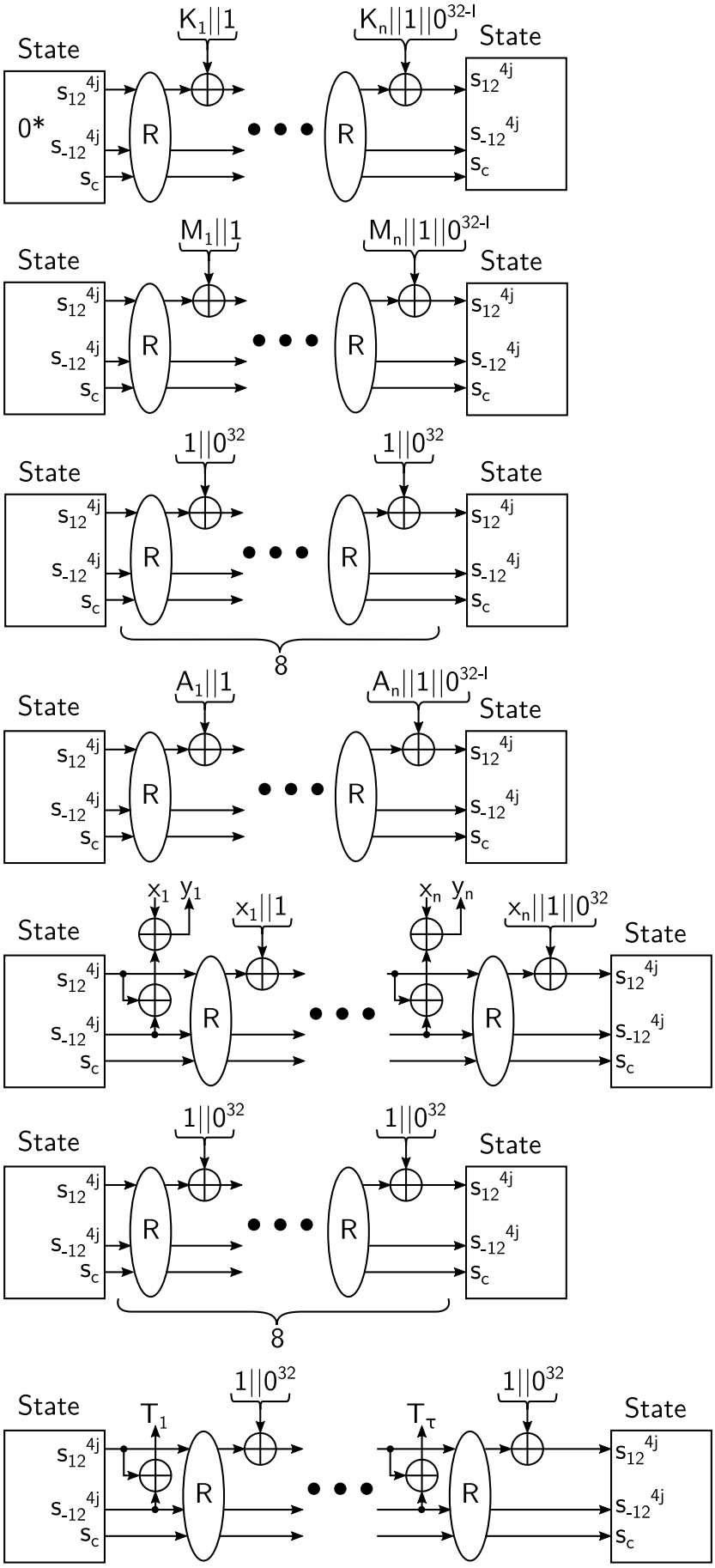

Figure 13: Subterranean-SAE, only encryption is illustrated.

E Hardware architecture with 4 rounds in one cycle

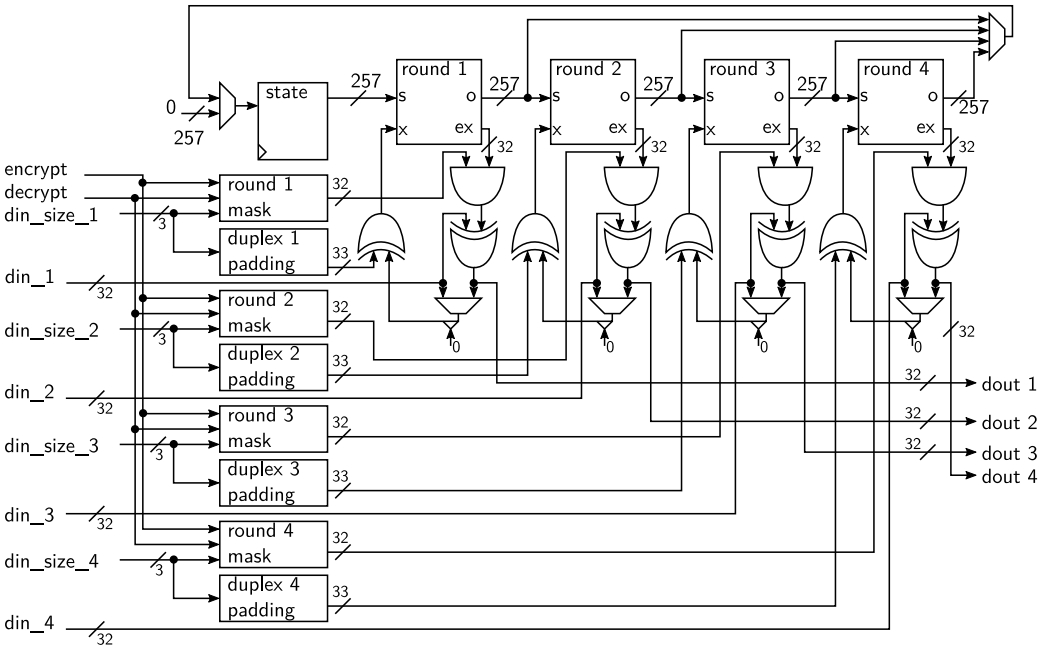

Figure 14: Subterranean 4-rounds duplex logic.