# Arithmetic Circuit Implementations of S-boxes for SKINNY and PHOTON in MPC⋆

Aysajan Abidin, Erik Pohle, and Bart Preneel

COSIC, KU Leuven, Belgium
`firstname.lastname@esat.kuleuven.be`

**Abstract.** Secure multi-party computation (MPC) enables multiple distrusting parties to compute a function while keeping their respective inputs private. In a threshold implementation of a symmetric primitive, e.g., of a block cipher, each party holds a share of the secret key or of the input block. The output block is computed without reconstructing the secret key. This enables the construction of distributed TPMs or transciphering for secure data transmission in/out of the MPC context. This paper investigates implementation approaches for the lightweight primitives SKINNY and PHOTON in arithmetic circuits. For these primitives, we identify arithmetic expressions for the S-box that result in smaller arithmetic circuits compared to the Boolean expressions from the literature. We validate the optimization using a generic actively secure MPC protocol and obtain 18% faster execution time with 49% less communication data for SKINNY-64-128 and 27% to 74% faster execution time with 49% to 81% less data for PHOTON $P_{100}$ and $P_{288}$. Furthermore, we find a new set of parameters for the heuristic method of polynomial decomposition, introduced by Coron, Roy and Vivek, specialized for SKINNY's 8-bit S-box. We reduce the multiplicative depth from 9 to 5.

**Keywords:** S-box · SKINNY · PHOTON · Secure Multi-Party Computation · Arithmetic Circuit

## 1 Introduction

Recent improvements in advanced cryptographic protocols, such as secure multi-party computation (MPC), fully homomorphic encryption (FHE), or zero-knowledge proof systems, made computation on encrypted data practical. This development enables privacy-preserving and GDPR compliant data processing and utilization in many areas, such as in public sector services, in smart cities, or healthcare. With added privacy benefits for users and data providers, various use cases emerge where cryptographic primitives are needed, including proofs over correct hashing, ciphertext-compression for FHE schemes, and secure outsourcing of computation and data storage for MPC. In this context, dedicated

PRFs [20, 25, 26], block and stream ciphers [1–3, 11, 19, 23, 37], and hash functions [22, 24] have been proposed that focus on minimizing multiplicative depth.

However, in a real-world scenario, cryptographic mechanisms and constructions need to interoperate between traditional computing systems (e.g., IoT devices, mobile phones, commodity and server CPUs) and these advanced cryptographic protocols. Traditional symmetric primitives, such as AES [38] and SHA-2 [39], are widely used in real-world applications and are widespread in internet and industry standards. For instance, proving properties of information obtained from real-world processes in zero-knowledge, e.g., the correct processing of financial transaction data, requires the usage of standardized constructions from that real-world domain since the information is not protected under non-standard cryptographic mechanisms. These standards almost exclusively specify traditional symmetric primitives at the core. Further, thresholdization of primitives, i.e., where the secret key is split among multiple parties who then jointly compute the relevant operation without reconstructing the secret key, is already deployed in industrial distributed key vault solutions and TPMs. Recently, NIST started investigating threshold implementations of NIST standardized primitives [10] as well. The important key part of thresholdization is that a threshold and a non-threshold implementation have to be interoperable, such that, e.g., derivative products of keys managed in a threshold fashion can be used in systems unaware of the thresholdization.

While thresholdized AES implementations have been studied, e.g., [13, 14, 17, 21, 31], other traditional primitives have not received that much attention. In this work, we want to study threshold implementations of lightweight primitives that may be used in applications where AES is undesirable. Lorünser and Wohner [35] implement several symmetric ciphers using two MPC frameworks, namely, MP-SPDZ and MPyC, to facilitate a better understanding of the two MPC frameworks. However, they treat the primitives as black boxes with little optimization of the primitive's performance. In a recent and more relevant work, motivated partly by the interoperability of privacy-enhancing protocols and lightweight cryptography, Mandal and Gong [36] study the Boolean circuit complexity of the core primitives of the NIST Lightweight Cryptography Competition (LWC)[1] round 2 candidates. However, their study is limited to Boolean circuits using the HalfGates garbling scheme by Zahur et al. [41] for the MPC (2PC to be more precise) evaluation of the ciphers.

To complement this effort, we move to the arithmetic circuit setting where variables are elements of, e.g., a finite field or ring, and basic gates are addition and multiplication gates. We investigate whether such a representation results in benefits, e.g., reduced circuit size, faster execution, or fewer communication data, over a straight-forward emulation of Boolean arithmetic paired with known Boolean circuits of lightweight primitives. A possible avenue in the arithmetic setting is to identify operations and structure in the primitive where groups of bits can be encoded as field/ring elements and equivalent arithmetic operations in the field/ring can replace bit-oriented functionality. For this purpose, we an-

---

[1] https://csrc.nist.gov/Projects/Lightweight-Cryptography

alyze the ten LWC finalists, but we limit our study to substitution-permutation network (SPN) designs of the underlying primitives. We exclude SPARKLE, Grain-128AEAD, and TinyJambu. Moreover, we rule out the permutations used in sponge-based AEADs (Ascon, ISAP, and Xoodyak) for two reasons. First, independent from the suitability of the permutation, the sponge structure creates highly serial circuits with high multiplicative depth that results in poor performance in non-constant round MPC protocols. Second, the permutation's round function operates over lanes, sheets, and columns of the state, mixing bits over all dimensions. This makes grouping bits within the state costly without a foreseeable benefit for arithmetic purposes. The SPN primitives of Elephant and GIFT-COFB involve a bit-level permutation making the linear layer costly (when grouped). Ultimately, we identify two primitives, SKINNY and PHOTON, stemming from the finalists Romulus and PHOTON-Beetle, respectively, where all operations on the state can be expressed as cell-wise operations and no intra-cell operations occur. We can therefore group the bits of each cell into one field/ring element and then investigate the cost of all operations in the arithmetic circuit. While we take SKINNY as the main demonstration example, all our findings also apply to a threshold implementation of PHOTON.

Our contributions can be summarized as follows:

- We provide several program representations for the SKINNY primitive in arithmetic circuits over $\mathbb{F}_{2^k}$ (see Sect. 3) optimized for usage in MPC protocols. We identify a trade-off between multiplications and pre-processed random bits for the evaluation of polynomials, resulting in a reduced number of multiplications for all 4-bit S-boxes.
- We benchmark the promising candidates of the trade-off in the secret sharing based "SPDZ-like" protocol MASCOT in the active security setting (see Sect. 4). We confirm the trade-off in practice and obtain improved performance for SKINNY variants with 64-bit block size, i.e., faster execution and lower communication cost, compared to the baseline.
- We show how the results for SKINNY carry over to a threshold implementation of PHOTON (see Sect. 4.3). We obtain similar performance improvements for 4-bit S-boxes and can apply well-known optimizations of the AES S-box used in the 8-bit PHOTON instance.

The rest of this paper is organized as follows. We give an introduction and background information on SKINNY, PHOTON and on the MPC protocol in Sect. 2. Then, we investigate the representation of SKINNY in arithmetic circuits in Sect. 3. The results of the experimental benchmark are detailed and discussed in Sect. 4. We conclude the paper in Sect. 5.

## 2 Background on Primitives and MPC

In the following, we give background details on the SKINNY lightweight block cipher family (Sect. 2.1), the permutations defined in PHOTON (Sect. 2.2) and discuss one MPC protocol for arithmetic circuits (Sect. 2.3).

## 2.1 SKINNY

SKINNY is a lightweight tweakable block cipher proposed by Beierle et al. [7], which has received a substantial amount of cryptanalysis (see, e.g., [5,18,28,34, 40,42] and [8, §5.3]). The primitive is designed to have a low hardware footprint, i.e., a small area on chip, and can therefore be used in IoT devices, embedded devices or RFID tags. Many implementations for hardware and software are available due to the NIST LWC process as part of the candidates Romulus [29], SKINNY-AEAD [8] and ForkAE [4][2].

SKINNY comes in several variants that process 64-bit or 128-bit blocks, and 64–384-bit tweakeys. A tweakey is the concatenation of a (secret), e.g., 64- or 128-bit key and a (public) tweak. The round function of this SPN cipher is similar to many AES-like primitives. Table 1 lists the number of rounds specified for each variant. The round function alters the internal state, a $4 \times 4$ array of $s$-bit cells. For a block size of 64-bit, $s = 4$, for 128-bit block size, $s = 8$. The initial state is the message block. Let the message be a sequence of $s$-bit values $s_0 \ s_1 \ \ldots \ s_{15}$, then the $4 \times 4$ array is filled row-wise:

$$\begin{pmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{pmatrix} \ .$$

The resulting ciphertext is the state after all rounds have been computed. The tweakeys are loaded into $4 \times 4$ arrays, TK1, TK2, TK3, in the same manner. TK1, present in all variants, is loaded with tweakey bits $0 \cdots (16s - 1)$. TK2 and TK3 are loaded with tweakey bits $16s \cdots (32s - 1)$ and $32s \cdots (48s - 1)$ respectively, if needed. The round function applies five steps in series: SubCells, AddRoundConstants, AddRoundKey, ShiftRows and MixColumns.

Table 1: The number of rounds for each variant of SKINNY. Variants are denoted by SKINNY-$b$-$tk$ where $b$ is the block size in bits and $tk$ is the tweakey size in bits. Note that the key size equals the block size in all variants.

| Variant | Block Size | Rounds | Variant | Block Size | Rounds |
|---|---|---|---|---|---|
| SKINNY-64-64 | | 32 | SKINNY-128-128 | | 40 |
| SKINNY-64-128 | 64 | 36 | SKINNY-128-256 | 128 | 48 |
| SKINNY-64-192 | | 40 | SKINNY-128-384 | | 56 |

**SubCells.** SubCells applies the S-box to each cell in the state. For $s = 4$, the 4-bit S-box is used (see Fig. 1a), for $s = 8$, the 8-bit S-box is used (see Fig. 1b).

---

Both S-boxes are computed by repeating XOR and NOR operations, and bit permutations. For the S-box definition as a truth table, we refer the reader to the original specification document [7].

**AddRoundConstants.** This step XORs public constants to three cells:

$$s_0' \leftarrow s_0 \oplus c_0 \ , \ \ s_4' \leftarrow s_4 \oplus c_1 \ , \ \ s_8' \leftarrow s_8 \oplus \texttt{0x2} \ .$$

The constants $c_0$ and $c_1$ are defined for each round, whereas the operand for $s_8$ remains $\texttt{0x2}$.

**AddRoundKey.** In each round, the first two rows of the state are XORed cell-wise with the first rows of each available round tweakey. Let $a_{i..j} \oplus b_{i..j}$ be a short-hand notation for $a_i \oplus b_i \ \ldots \ a_j \oplus b_j$, then

$$s_{0..3}' \leftarrow s_{0..3} \oplus \mathrm{TK1}_{0..3} \oplus \mathrm{TK2}_{0..3} \oplus \mathrm{TK3}_{0..3} \ ,$$
$$s_{4..7}' \leftarrow s_{4..7}' \oplus \mathrm{TK1}_{4..7} \oplus \mathrm{TK2}_{4..7} \oplus \mathrm{TK3}_{4..7} \ .$$

**ShiftRows.** Shift rows applies a cell-wise permutation $P_S$ on the state where

$$P_S(0,...,15) = (0,1,2,3,7,4,5,6,10,11,8,9,13,14,15,12) \ .$$

This rotates each row by $0, 1, 2$ and $3$ elements to the right.

**MixColumns.** The MixColumns step multiplies the state with the matrix

$$\begin{pmatrix} s_0' & s_1' & s_2' & s_3' \\ s_4' & s_5' & s_6' & s_7' \\ s_8' & s_9' & s_{10}' & s_{11}' \\ s_{12}' & s_{13}' & s_{14}' & s_{15}' \end{pmatrix} \leftarrow \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{pmatrix} \ .$$

**Key Schedule.** The key schedule describes how a round key is derived from the cipher's key. The first round key is the tweakey itself. Round keys for subsequent rounds are obtained by applying the permutation $P_T$ cell-wise on the $4 \times 4$ array representation of each tweakey. Each cell in TK2 and TK3 is further updated by a linear feedback shift register (LFSR). In short, denoting the round key for the next round by $\mathrm{TK}i'$, $i = 1, 2, 3$, we have

$$\mathrm{TK1}' \leftarrow P_T(\mathrm{TK1}), \ \ \mathrm{TK2}' \leftarrow \ \ \mathrm{LFSR2} \circ P_T(\mathrm{TK2}), \ \ \mathrm{TK3}' \leftarrow \mathrm{LFSR3} \circ P_T(\mathrm{TK3}),$$

where

$$P_T(0,..,15) = (9,15,8,13,10,14,12,11,0,1,2,3,4,5,6,7)$$

and LFSR2/LFSR3 are defined in Table 2. $P_T$ swaps the first two rows with the last two rows of the state and applies a permutation to the now first two rows.

Table 2: Linear feedback shift registers LFSR2 and LFSR3 defined in the key schedule of SKINNY for tweakeys TK2 and TK3, respectively.

|  | Cell size |  |  |
|---|---|---|---|
| LFSR2 | 4-bit | $(x_3, x_2, x_1, x_0)$ | $\rightarrow (x_2, x_1, x_0, x_3 \oplus x_2)$ |
|  | 8-bit | $(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0)$ | $\rightarrow (x_6, x_5, x_4, x_3, x_2, x_1, x_0, x_7 \oplus x_5)$ |
| LFSR3 | 4-bit | $(x_3, x_2, x_1, x_0)$ | $\rightarrow (x_0 \oplus x_3, x_3, x_2, x_1)$ |
|  | 8-bit | $(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0)$ | $\rightarrow (x_0 \oplus x_6, x_7, x_6, x_5, x_4, x_3, x_2, x_1)$ |

## 2.2 PHOTON

We briefly describe the internal permutations $P_t$, $t \in \{100, 144, 196, 256, 288\}$, of the PHOTON hash function [27]. Similarly to SKINNY, the internal state is a $d \times d$ array of $s$-bit cells that is transformed by applying the following round function steps 12 times: AddConstant, SubCells, ShiftRows, MixColumnsSerial. Since $P_t$ is a permutation, it has no secret key addition layer. Table 3 lists the parameters for each variant.

**AddConstant.** Public round constants and instance-specific internal constants are XORed to the first column of the state.

**SubCells.** If $s = 4$, the PRESENT S-box [9] is applied to each cell in the state. If $s = 8$, the AES S-box is applied.

**ShiftRows.** This applies a cell-wise permutation on the state where row $i$ is rotated by $i$ columns to the left.

**MixColumnsSerial.** Each column of the state is multiplied with a matrix $A_t$ $d$ times. The multiplication is defined over $\mathbb{F}_2[X]/X^4 + X + 1$ for $s = 4$ and over $\mathbb{F}_2[X]/X^8 + X^4 + X^3 + X + 1$ for $s = 8$.

Table 3: State size $d$, cell size $s$ and modulus of PHOTON $P_t$.

| Instance | $d$ | $s$ | Modulus |
|---|---|---|---|
| $P_{100}$ | 5 | 4 | |
| $P_{144}$ | 6 | 4 | $X^4 + X + 1$ |
| $P_{196}$ | 7 | 4 | |
| $P_{256}$ | 8 | 4 | |
| $P_{288}$ | 6 | 8 | $X^8 + X^4 + X^3 + X + 1$ |

## 2.3 A Multi-Party Computation Protocol for Arithmetic Circuits

In this and the following sections, we denote a uniform random sampling from a finite set $A$ with $\overset{\$}{\leftarrow} A$. We now briefly discuss the SPDZ-style, dishonest-

majority MPC protocol on arithmetic circuits that achieves active security using information-theoretically secure MACs. The communication model in the protocol assumes secure point-to-point channels and a synchronous network. If we later refer to a round of communication, this means each party broadcasts one or more local values to all other parties. In this model, the broadcast based on point-to-point connections costs $\mathcal{O}(n^2)$ values to send for $n$ players. In the protocol, the computation is split into a pre-processing, a.k.a. offline, phase and an online phase. In the offline phase, the players jointly create correlated randomness for multiplication and bit-decomposition. Since neither the individual party's inputs nor the concrete function to compute[3] have to be known, this phase can take place well before the online phase and is usually computationally much heavier. In the online phase, the parties know their own inputs and the arithmetic circuit. This phase consumes the correlated randomness from the offline phase. Since we only consider binary extension fields in this paper, we adapt the notation for the MPC protocol accordingly. Recall that $\mathbb{F}_{2^k} = \mathbb{F}_2[X]/Q(X)$ is a finite field with $2^k$ elements, where $k > 0$. Each element can be represented as a polynomial of degree at most $k-1$ whose coefficients are in $\mathbb{F}_2$ and $Q(X)$ is an irreducible polynomial of degree $k$. Addition $g(X)+h(X)$, for $g(X), h(X) \in \mathbb{F}_{2^k}$, is performed coefficient-wise. Multiplication $g(X)h(X)$ is the ordinary polynomial multiplication modulo $Q(X)$. Every variable in the arithmetic circuit is an element in $\mathbb{F}_{2^k}$. During execution, each player holds or obtains an additive secret share of every variable. We denote the additive share of $x \in \mathbb{F}_{2^k}$ of player $i$ with $x^{(i)}$, i.e., $\sum x^{(i)} = x$. A SPDZ-like share of the same player is denoted with $[\![x]\!]_i = \langle x^{(i)}, m^{(i)} \rangle$ which carries a MAC share $m^{(i)}$ that authenticates the secret share to enable active security where $m$ is created using the global secret MAC key $\Delta \in \mathbb{F}_{2^k}$.

**Offline Phase.** The offline phase implements the functionalities $\mathcal{F}_{\mathsf{Triple}}$ and $\mathcal{F}_{\mathsf{Bit}}$ by using somewhat homomorphic encryption SHE (e.g. in [15,16,33]) or oblivious transfer [32]. While the offline phase dominates the total runtime of the MPC protocol, its details are less important for the purpose of this paper. We invite the reader to consult the aforementioned references for further details.

The functionality $\mathcal{F}_{\mathsf{Triple}}$ produces Beaver multiplication triples [6] of the form $([\![a]\!], [\![b]\!], [\![c]\!])$ where $c = ab$ and $a, b \xleftarrow{\$} \mathbb{F}_{2^k}$. The functionality $\mathcal{F}_{\mathsf{Bit}}$ produces random bits $[\![r]\!]$ with $r \xleftarrow{\$} \{0, 1\}$.

**Online Phase.** Before detailing the addition and multiplication of shares, we have to describe the concept of (partially) opening a share. In general, if a share $[\![x]\!]$ is opened, each player $i$ broadcasts $x^{(i)}$ and then sums up all shares to obtain $x$. For active security, the players first commit to the MAC shares $m^{(i)} - \Delta^{(i)} x^{(i)}$ before opening them. Later it is checked whether $m - \Delta x = 0$. The core idea of SPDZ is to defer the checking of the MAC values to the very end of the protocol,

---

[3] However, the players must know an upper bound on the number of required multiplication triples resp. random bits.

resulting in a so-called partial open. Before the final output is revealed, all MACs of partially opened shares are checked in one go. If this check passes, the output value is reconstructed.

Let $[\![x]\!] = \langle x^{(i)}, m_x^{(i)} \rangle, [\![y]\!] = \langle y^{(i)}, m_y^{(i)} \rangle$ be shares and $e \in \mathbb{F}_{2^k}$ a public constant, then addition of shares, public constants and multiplication by public constants can be performed locally by each player:

$$e + [\![x]\!] = [\![e + x]\!] : \begin{cases} \langle x^{(0)} + e, m_x^{(0)} + e\Delta^{(0)} \rangle & \text{if } i = 0 \,, \\ \langle x^{(i)}, m_x^{(i)} + e\Delta^{(i)} \rangle & \text{else} \,, \end{cases}$$

$$e \cdot [\![x]\!] = [\![e \cdot x]\!] : \langle e \cdot x^{(i)}, e \cdot m_x^{(i)} \rangle \,,$$

$$[\![x]\!] + [\![y]\!] = [\![x + y]\!] : \langle x^{(i)} + y^{(i)}, m_x^{(i)} + m_y^{(i)} \rangle \,.$$

Given a multiplication triple $([\![a]\!], [\![b]\!], [\![c]\!])$ from $\mathcal{F}_{\mathsf{Triple}}$, we compute the multiplication $[\![x]\!] \cdot [\![y]\!] = [\![x \cdot y]\!]$ in two steps.

1. The players partially open $[\![x - a]\!]$ as $\gamma$ and $[\![y - b]\!]$ as $\epsilon$.
2. Each player computes locally $[\![x \cdot y]\!]_i = [\![c]\!]_i + \gamma \cdot [\![b]\!]_i + \epsilon \cdot [\![a]\!]_i + \gamma \cdot \epsilon$.

The partial open requires one round of communication, unlike the linear operations mentioned before.

We can also compute a bit-decomposition of a shared $x \in \mathbb{F}_{2^k}$ into $k$ shares of the bits of $x$, $b_0, \ldots, b_{k-1}$ where $x = \sum_{j=0}^{k-1} b_j X^j$. Note that the resulting bit $b_i$ is still shared over $\mathbb{F}_{2^k}$. Given $k$ random bits $[\![r_0]\!], \ldots, [\![r_{k-1}]\!]$ from $\mathcal{F}_{\mathsf{Bit}}$,

1. The players locally compute $[\![r]\!] = \sum_{j=0}^{k-1} [\![r_j]\!] X^j$ and partially open $[\![x - r]\!]$ as $\gamma$.
2. Let $\gamma_0, \ldots, \gamma_{k-1} \in \{0, 1\}$ be the (clear text) decomposition of $\gamma$. Each player then computes $[\![b_0]\!] = [\![\gamma_0 + r_0]\!], \ldots, [\![b_{k-1}]\!] = [\![\gamma_{k-1} + r_{k-1}]\!]$.

In summary, multiplying two secret-shared values, i.e., $[\![x \cdot y]\!] \leftarrow [\![x]\!] \cdot [\![y]\!]$, requires one multiplication triple from $\mathcal{F}_{\mathsf{Triple}}$ and one round of communication. A bit-decomposition of $[\![x]\!]$ into $k$ bits $[\![b_0]\!], \ldots, [\![b_{k-1}]\!]$ requires $k$ random bits from $\mathcal{F}_{\mathsf{Bit}}$ and one round of communication. Note that both for multiplication and bit-decomposition, data of independent operations can be sent in the same round.

## 3   Arithmetic Circuit Implementation

We aim to explore possible performance gains of an arithmetic representation of the circuit where we utilize properties of the underlying field over an emulation of Boolean arithmetic. Thus in the following, variables are elements of a finite field. The cell-focused nature of SKINNY allows the representation of each cell as a finite field element. Thus, the state consists of 16 field elements.

Concretely, we define two fields[4] of size $2^4$ and $2^8$,

$$\mathbb{F}_{2^4} = \mathbb{F}_2[X]/(X^4 + X^3 + 1) \,,$$

$$\mathbb{F}_{2^8} = \mathbb{F}_2[X]/(X^8 + X^7 + X^6 + X^5 + X^4 + X^2 + 1) \,. \tag{1}$$

---

[4] Since the SKINNY reference does not specify operations in a field, we are free to pick a suitable one.

For SKINNY versions with a 64-bit state, we pick the field $\mathbb{F}_{2^4}$ and for a 128-bit state, we use $\mathbb{F}_{2^8}$. We encode $s$-bit cell values $b_{s-1}\ldots b_0$ into field elements as $b_{s-1}\ldots b_0 \leftrightarrow \sum_{i=0}^{s-1} b_i X^i$. We express values from this correspondence as hexadecimal literals, e.g., `0xa3` $\leftrightarrow X^7 + X^5 + X + 1$. With this correspondence, XOR of two $s$-bit values translates to addition of two field elements in $\mathbb{F}_{2^s}$. As a result, all parts of the round function except for SubCells become linear and can be computed locally by each player. The fields defined in Eqn. (1) entail a minimal number of multiplications to implement the respective S-box via polynomial interpolation. We give more details later in Sect. 3.2. From Table 2, we can see that if the tweakey is available in shared bits, the LFSR computation, and thus the whole key schedule, is also linear and incurs no communication rounds.

Furthermore, we recall that squaring is a linear operation in fields of characteristic two, i.e.,

$$\left(\sum_{i=0}^{s-1} b_i X^i\right)^2 = \sum_{i=0}^{s-1} (b_i X^i)^2 . \tag{2}$$

Thus, given the bit-decomposition of $x \in \mathbb{F}_{2^s}$, any power of the form $x^{2^j}$ can be computed without any multiplication triples. We stress, however, that the initial bit-decomposition requires one opening in the online phase, so computing any number of squares in $\{x^2, x^4, x^8, ...\}$ costs one round of communication and $s$ random bits. Appendix A.1 gives an example of squaring as a linear operation.

In the following, we describe approaches to express the non-linear part of SubCells, the S-box. Section 3.1 describes the baseline approach that emulates Boolean arithmetic. Then, we study approaches via polynomial interpolation. Section 3.2 details the interpolation and Sect. 3.3 improves the evaluation by utilizing the free squaring property. In Sect. 3.4, we apply a polynomial decomposition to compute the S-box. Table 4 lists the cost of each S-box implementation approach in terms of multiplication triples, random bits and communication rounds.

## 3.1 Binary S-box

The Boolean operations AND, XOR and NOT can be naturally emulated in any field with characteristic two. If the values are a sharing of 0 or 1, AND is expressed as multiplication, XOR as addition and NOT is addition with the constant `0x1`. In this approach, each bit in an $s$-bit cell is encoded as a field element and we compute the S-box as given in the SKINNY specification [7] emulating Boolean operations (see Fig. 1). We will further use this approach as baseline for the comparison.

$$x_0' \leftarrow x_1 \oplus (\neg x_3' \wedge \neg x_2')$$
$$x_1' \leftarrow x_2 \oplus (\neg x_1 \wedge \neg x_3')$$
$$x_2' \leftarrow x_3 \oplus (\neg x_2 \wedge \neg x_1)$$
$$x_3' \leftarrow x_0 \oplus (\neg x_3 \wedge \neg x_2)$$

$$x_0' \leftarrow x_2 \oplus (\neg x_3' \wedge \neg x_1')$$
$$x_1' \leftarrow x_7 \oplus (\neg x_7' \wedge \neg x_2')$$
$$x_2' \leftarrow x_6 \oplus (\neg x_2 \wedge \neg x_1)$$
$$x_3' \leftarrow x_1 \oplus (\neg x_5' \wedge \neg x_3)$$
$$x_4' \leftarrow x_3 \oplus (\neg x_7' \wedge \neg x_6')$$
$$x_5' \leftarrow x_0 \oplus (\neg x_3 \wedge \neg x_2)$$
$$x_6' \leftarrow x_4 \oplus (\neg x_7 \wedge \neg x_6)$$
$$x_7' \leftarrow x_5 \oplus (\neg x_6' \wedge \neg x_5')$$

(a) The 4-bit S-box.          (b) The 8-bit S-box.

Fig. 1: The 4-bit and 8-bit S-box of the SKINNY cipher. The cell bit $x_i$ is transformed into $x_i'$.

### 3.2 S-box via Polynomial Interpolation

Another representation of the ($s$-bit) S-box is via a polynomial $P_s(z) = \sum_{i=0}^{2^s-1} a_i z^i$, where $a_i \in \mathbb{F}_{2^s}$. Then, the computation of the S-box on a given value $x$ is the evaluation of $P_s$ at $x$. We can obtain the coefficients $a_i$ by associating $(x, \mathcal{S}_s(x))$ for all $x \in \mathbb{F}_{2^s}$ and computing the interpolating polynomial by means of Lagrange interpolation, or by solving the following linear system of equations

$$\begin{pmatrix} \mathsf{0x1} \ \mathsf{0x0}^1 \ \dots \ \mathsf{0x0}^{2^s-1} \\ \mathsf{0x1}^0 \ \mathsf{0x1}^1 \ \dots \ \mathsf{0x1}^{2^s-1} \\ \vdots \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_{2^s-1} \end{pmatrix} = \begin{pmatrix} \mathcal{S}_s(\mathsf{0x0}) \\ \mathcal{S}_s(\mathsf{0x1}) \\ \vdots \end{pmatrix} .$$

This approach primarily motivated the choice for the irreducible polynomials in Eqn. (1). The chosen modulus entails a maximally sparse interpolating polynomial for the respective S-box, i.e., for this modulus, $P_s(z)$ contains the maximal number of coefficients $a_i = \mathsf{0x0}$.

The interpolating polynomial for SKINNY's 4-bit S-box $\mathcal{S}_4$ is

$$P_4(z) = \mathsf{0xc} + \mathsf{0x8}z + \mathsf{0x3}z^2 + \mathsf{0xd}z^3 + \mathsf{0xf}z^4 + \mathsf{0x4}z^5 + \mathsf{0x8}z^6 + \mathsf{0x6}z^7$$
$$+ \mathsf{0x1}z^8 + \mathsf{0x9}z^9 + \mathsf{0x8}z^{10} + \mathsf{0xe}z^{12} + \mathsf{0xc}z^{13} + \mathsf{0xb}z^{14} . \qquad (3)$$

The inverse $\mathcal{S}_4^{-1}$ is slightly sparser, with one less non-zero coefficient. For the 8-bit S-box $\mathcal{S}_8$, $P_8(z)$ is more unwieldy with degree 252 and 244 non-zero coefficients. Its inverse $\mathcal{S}_8^{-1}$ has degree 252 with 241 non-zero coefficients.

For a direct evaluation of $P(z)$, we need to compute the powers $z^i$ that occur in $P(z)$. The remaining linear combination $\sum a_i z^i$ is free. In order to minimize the number of sequential multiplications, we express the computation through the shortest addition chain of the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14\}$ for $P_4$ (see Fig. 2a). This approach is marked as MUL in Table 4. Analogously for $\mathcal{S}_8$, we

find a chain that requires 242 multiplications in 8 rounds and for $\mathcal{S}_8^{-1}$, we use 239 multiplications in 8 rounds.



(a) Only via multiplications (MUL).

(b) Using free squares $2, 4, 8$ (SQ1). Dashed arrows denote free squaring via bit-decomposition.
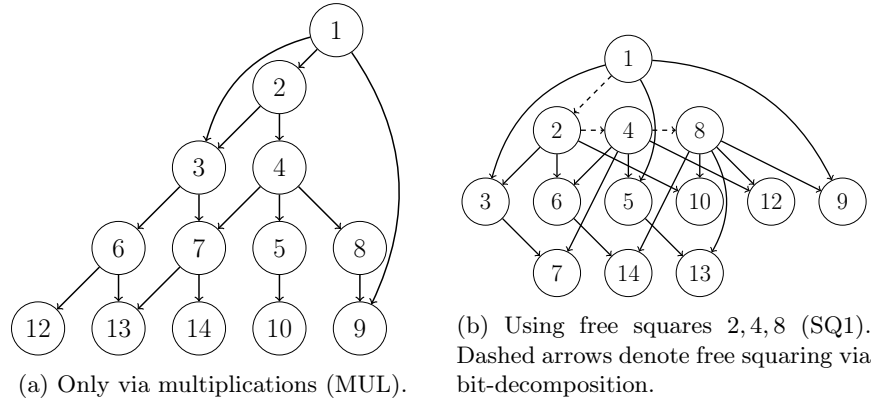
Fig. 2: Shortest addition chain for powers in the interpolating polynomial for SKINNY's 4-bit S-box. Each level in the tree denotes one communication round.

### 3.3 S-box via Polynomial Interpolation with Free Squaring

We may use bit-decomposition and then repeated free squaring to compute more powers in a single round. This creates a trade-off between multiplicative depth, the number of multiplications and the number of required pre-processed random bits for the bit-decomposition. We explore this trade-off for the 4-bit S-box in detail since the number of powers to compute is significantly smaller than for the 8-bit S-box. We denote this approach SQ1, SQ2, ... where one, two, ... base values are used for free squaring. Table 4 lists the cost for each combination. For $\mathcal{S}_4$ and SQ1, we first square $z^1$ to obtain $z^2, z^4, z^8$. This is illustrated in Fig. 2b. For SQ2, we compute $z^3$ normally and also square it to obtain $z^6, z^{12}, z^9$ for free. For SQ3, $z^5$ is squared to obtain $z^{10}$ and for SQ4 squaring $z^7$ yields $z^{14}, z^{13}, z^{11}$. While squaring once/twice, e.g., SQ1 and SQ2, decreases the number of rounds that are necessary for the computation, SQ3 and SQ4 require one more round. The reason for the additional required round is that some powers can no longer be computed in the original round since the prerequisite powers are no longer both available in the previous round because they are computed later for free. Concretely, power 14 can no longer be computed in round 3 by using powers 6 and 8 since power 6 is computed for free at the earliest in round 3. Figure 5a in Appendix A.2 illustrates this by showing the addition chain for SQ3.

We visualize the trade-off in the 8-bit case in Fig. 5b in Appendix A.2. Three configurations may be of interest. The plain multiplication approach requires 242 multiplications in 8 rounds but no random bits. Using only the square chain $1 \to 2 \to 4 \to 8 \to \ldots$ requires 236 multiplications, 8 random bits in 4 rounds.

On the other end, if as many values are computed via squaring as possible, the computation requires 33 multiplications and 264 random bits in 5 rounds.

Table 4: Cost of implementation approaches for SKINNY's 4-bit and 8-bit S-boxes. MUL denotes the direct evaluation of the interpolating polynomial, BIN is the emulation of Boolean arithmetic, SQ$i$ denotes utilization of $i$ free square chains and CRV denotes the polynomial decomposition.

|  | $\mathcal{S}_4$ | | | $\mathcal{S}_4^{-1}$ | | |
|  | Mult. | Bits | Depth | Mult. | Bits | Depth |
|---|---|---|---|---|---|---|
| MUL | 12 | 0 | 4 | 11 | 0 | 4 |
| BIN | 4 | 0 | 2 | 4 | 0 | 4 |
| SQ1 | 9 | 4 | 3 | 8 | 4 | 3 |
| SQ2 | 6 | 8 | 3 | 6 | 8 | 3 |
| SQ3 | 5 | 12 | 4 | 5 | 12 | 4 |
| SQ4 | 3 | 16 | 4 | 3 | 16 | 4 |
| CRV | 2 | 8 | 4 | 2 | 8 | 4 |

|  | $\mathcal{S}_8$ | | | $\mathcal{S}_8^{-1}$ | | |
|  | Mult. | Bits | Depth | Mult. | Bits | Depth |
|---|---|---|---|---|---|---|
| MUL | 242 | 0 | 8 | 239 | 0 | 8 |
| BIN | 8 | 0 | 4 | 8 | 0 | 4 |
| SQ1 | 236 | 8 | 4 | 233 | 8 | 4 |
| SQ33 | 33 | 264 | 5 | 32 | 256 | 5 |
| CRV | 10 | 40 | 5 | 10 | 40 | 5 |

### 3.4 Decomposition

We can use the decomposition method, CRV, by Coron, Roy and Vivek [12] to reduce the number of multiplications to evaluate the interpolating polynomial $P(z)$. In short, $P(z)$ is decomposed into the sum of products of polynomials $p_i(z)$ and $q_i(z)$,

$$P(z) = \sum_{i=1}^{t-1} p_i(z)q_i(z) + p_t(z) \ , \tag{4}$$

where each polynomial $p_i$, $q_i$ only has monomials $z^a$ with $a \in L$ (see below). The set $L$ is constructed from a number of cyclotomic bases $\alpha_j$, where each base constructs the consecutive squares starting from $\alpha_j$

$$C_{\alpha_j} = \{2^i \alpha_j \mod 2^s - 1 \mid \forall 0 \le i < 2^s\} \tag{5}$$

and thus

$$L = C_{\alpha_1} \cup \cdots \cup C_{\alpha_l} . \tag{6}$$

With a good choice of $l$ cyclotomic bases, all powers $z^a$ for $a \in L$ can be computed with $l - 2$ multiplications. Naturally, $\alpha_1 = 0$ and $\alpha_2 = 1$, i.e., $z^0$ and $z^1$, which don't require any computation. Essentially, $z^{\alpha_j}$ is computed as the product of previous values, while $z^{2^i \alpha_j}$ is computed for free since squaring is linear in our chosen field. Therefore, the entire polynomial can be evaluated with $l - 2 + t - 1$ multiplications by first computing the monomials defined by $L$ and then computing the product $p_i(z)q_i(z)$.

Table 5: Parameter choices for the polynomial decomposition in $\mathbb{F}_{2^s}$ and the evaluation cost in terms of multiplication triples, random bits and multiplicative depth. The parameter $t$ denotes the number of $p_i/q_i$ polynomials in Eqn. (4).

| | $s$ $t$ | Base $\alpha$ | Mult. | Bits | Depth |
|---|---|---|---|---|---|
| CRV [12] | 4 2 | $\{0, 1, 3\}$ | 2 | 8 | 4 |
| CRV [12] | 8 6 | $\{0, 1, 3, 7, 29, 87, 251\}$ | 10 | 48 | 9 |
| Ours for $\mathcal{S}_8$ and $\mathcal{S}_8^{-1}$ | 8 7 | $\{0, 1, 3, 5, 7, 11\}$ | 10 | 40 | 5 |

The CRV method is heuristic as one chooses the cyclotomic bases and co-efficients for polynomials $q_i$ to solve the resulting linear system for coefficients of $p_i$. The authors of [12] give $\alpha$ values for 4- and 8-bit polynomials for which random choices for $q_i$ lead to a system with a solution.

Their parameter choice was motivated by finding higher-order masking to protect implementations against side-channel attacks and has a minimal number of multiplications. For our scenario, we also attempt to reduce the multiplicative depth since this reduces the number of communication rounds in the protocol. Table 5 lists our parameter choice and the heuristics given in [12]. For the 4-bit case, the choice $\alpha_j \in \{0, 1, 3\}$ is also minimal in terms of communication rounds. For the specific S-boxes $\mathcal{S}_8$ and $\mathcal{S}_8^{-1}$, we find a new set of cyclotomic bases with a lower multiplicative depth and less random bits which only increases the number of linear operations. For the 4-bit case, we detail the coefficients of $p_i/q_i$ in Table 7 in Appendix A.3.

Using this approach, *any* 4-bit S-box can be implemented requiring 2 multiplications and 8 random bits in 4 rounds. Our new parameters implement SKINNY's 8-bit S-boxes with 10 multiplications and 40 random bits in 5 rounds, however, they don't allow the implementation of any 8-bit S-box[5].

## 4 Experimental Results

We implemented two cipher variants, SKINNY-64-128 and SKINNY-128-256, in the forward and inverse direction. In Sect. 4.1 we evaluate all S-box approaches for SKINNY's 4-bit S-box and in Sect. 4.2, we investigate the BIN and CRV variant for SKINNY's 8-bit S-box. Finally, we apply the results to PHOTON in Sect. 4.3. Table 6 shows the gate counts for the complete primitives. In all comparisons, BIN denotes the baseline.

We benchmark in a three-party LAN setting[6] using the MASCOT MPC protocol [32] in the MP-SPDZ framework [30]. MASCOT provides active security for a dishonest majority. In the MP-SPDZ implementation, shares are elements of the field $\mathbb{F}_{2^{40}}$ defined as $\mathbb{F}_{2^{40}} = \mathbb{F}_2[Y]/(Y^{40}+Y^{20}+Y^{15}+Y^{10}+1)$. We therefore

---

[5] The parameters cannot be used to decompose the AES S-box, for instance.

[6] Each party runs on a separate machine with 4 cores and 16 GB RAM connected with a bandwidth of 10 Gbit/sec and <1 ms latency.

Table 6: Gate counts of SKINNY-64-128, SKINNY-128-256, PHOTON $P_{100}$, PHOTON $P_{288}$ and AES-128 (for context). Add/Cmul denote the number of local linear operations.

| | Mult. | Random Bits | Add/Cmul | Comm. Rounds |
|---|---|---|---|---|
| SKINNY-64-128 (BIN) | 2304 | 0 | 10238 | 72 |
| SKINNY-64-128 (CRV) | 1152 | 4608 | 82764 | 144 |
| SKINNY-128-256 (BIN) | 6144 | 0 | 27465 | 145 |
| SKINNY-128-256 (CRV) | 7680 | 30720 | 1545744 | 240 |
| PHOTON $P_{100}$ (BIN) | 1200 | 0 | 13862 | 48 |
| PHOTON $P_{100}$ (CRV) | 600 | 2400 | 56520 | 48 |
| PHOTON $P_{288}$ (BIN) | 13824 | 0 | 135648 | 72 |
| PHOTON $P_{288}$ (AES) | 2592 | 6912 | 207072 | 60 |
| AES-128 [14] | 1200 | 3200 | 45149 | 53 |

embed both $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^8}$ into $\mathbb{F}_{2^{40}}$. This also achieves 40-bit statistical security. Let $\mathcal{E}_4$ and $\mathcal{E}_8$ denote the embedding $\mathbb{F}_{2^4} \hookrightarrow \mathbb{F}_{2^{40}}$ and $\mathbb{F}_{2^8} \hookrightarrow \mathbb{F}_{2^{40}}$, respectively. We use $\mathcal{E}_4(Y) = Y^{35} + Y^{20} + Y^5 + 1$ and $\mathcal{E}_8(Y) = Y^{35} + Y^{30} + Y^{25} + Y^{20} + Y^{10} + Y^5$ as they require the lowest number of linear operations to be computed among all available embeddings. Note that decomposing an embedded element from $\mathbb{F}_{2^s}$ still only costs $s$ random bits (see Appendix A.4 for more details).

We compute 100 circuits (key schedule, if applicable, and block encryption/decryption) in parallel to allow for amortization effects in the pre-processing phase. Both the input block and the key are secret inputs and not entirely known by any party. Note that if one party fully knows the key, it may be more efficient to compute the key schedule locally and input each round key separately. We compute the key schedule within the MPC protocol to make our experiments more broadly usable, if, e.g., the key is the result of a previous MPC computation or each party inputs a key share. Our code, scripts and artifacts are available online[7].

## 4.1 SKINNY-64-128

We choose the SKINNY-64-128 variant to assess the performance of all 4-bit S-box implementation approaches. Any performance gains for SKINNY-64-64 or SKINNY-64-192 will be similar since these variants only differ in the number of rounds and the linear key schedule.

Figure 3a visualizes the total, i.e., pre-processing and online, runtime and total communication data per player per encryption/decryption and S-box implementation approach for SKINNY-64-128. We note that the number of multiplications in the circuit seems to dominate the total performance regarding time and data. The more free squares are used, the lower the time and data.

---

[7] https://github.com/ErikP0/arithmetic-circuits-for-spn-primitives

While the SQ4 approach uses fewer multiplications than BIN, we measure fewer data but a slower total time, presumably due to the two additional rounds and four bit-decompositions. The CRV implementation performs best in time and data compared to all other approaches, including the baseline Boolean arithmetic emulation BIN. At least in our setting, trading-off two multiplications with two bit-decompositions (and thus eight random bits) leads to better overall performance. SQ4 is around 24% slower but uses 23% less data than BIN. CRV is approx. 18% faster and uses 49% less data than BIN.



| | |
|---|---|
| o,× | MUL |
| o,× | SQ1 |
| o,× | SQ2 |
| o,× | SQ3 |
| o,× | SQ4 |
| o,× | BIN |
| o,× | CRV |
| o,× | AES-128 |

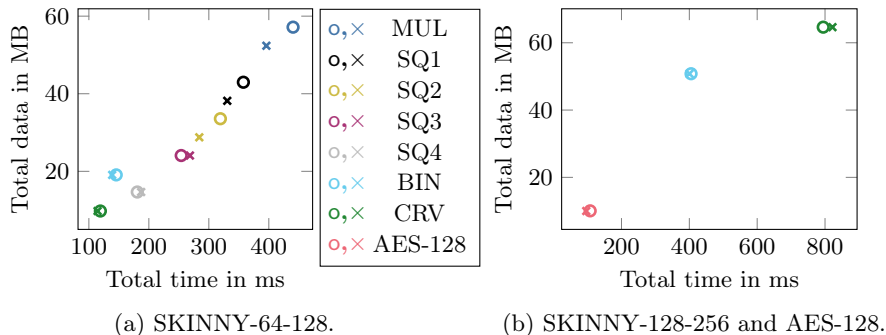(a) SKINNY-64-128.  (b) SKINNY-128-256 and AES-128.

Fig. 3: Total, i.e., pre-processing and online, execution time and communication data for multiple S-box implementation approaches of SKINNY-64-128 and SKINNY-128-256 amortized with 100 executions in parallel. The legend symbol **o** denotes the forward direction while × denotes the inverse direction.

## 4.2  SKINNY-128-256

We implemented the BIN and CRV approach for the 8-bit S-boxes since the MUL or SQ1/SQ33 approaches are not better than CRV or BIN in any metric, i.e., number of multiplications, number of random bits or multiplicative depth. We evaluate BIN and CRV in SKINNY-128-256 and report the total time and communication data per player in Fig. 3b. In the same figure, we also give total time and communication data of an AES forward and inverse computation in the same setting following the implementation from Damgård et al. [14].

As already visible in the gate counts (cf. Table 4), the CRV approach does not create a favourable trade-off for the 8-bit S-box. This means that the BIN baseline approach is faster and uses less data than CRV. Furthermore, for the block size of 128 bits, AES outperforms SKINNY-128-256. The S-box of AES is much cheaper to implement arithmetically, via 6 multiplications and two bit-decompositions than the Boolean implementation that would require 32 multiplications. In addition, AES only has ten rounds while SKINNY-128-256 has more than four times more rounds.

### 4.3 PHOTON

Finally, we transferred the results to PHOTON. The four defined permutations $P_{100}, P_{144}, P_{196}$ and $P_{256}$ use the 4-bit S-box of PRESENT [9] while $P_{288}$ uses the AES S-box. The PHOTON permutations have mixing layers where the state is multiplied with a mixing matrix in a pre-defined finite field. While it may seem that this complicates the implementation approaches, a fixed modulus is not a problem since the CRV method (for the 4-bit case) applies to any field with the same cost. Further, any AES S-box implementation may be applied to $P_{288}$. To illustrate how our results carry over, we implemented $P_{100}$ and $P_{288}$. For $P_{100}$, we apply the CRV decomposition approach, and for $P_{288}$ we apply the known AES S-box optimizations from [14].

Figure 4 illustrates the benchmark results. For $P_{100}$, we note a 27% faster execution with 49% less data. For $P_{288}$, we observe a 74% faster execution with 81% less data.
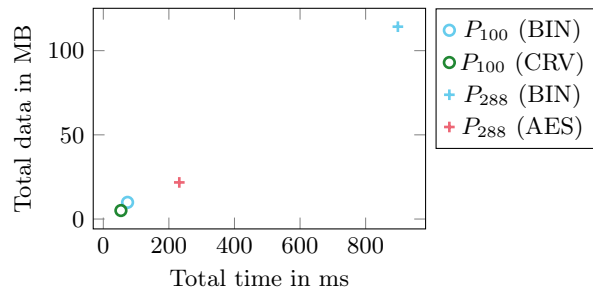


Fig. 4: Total, i.e., pre-processing and online, execution time and communication data for PHOTON $P_{100}$ and $P_{288}$ amortized with 100 executions in parallel.

## 5   Conclusion

We investigated and identified improvements of an arithmetic circuit representation of the most costly component of the SKINNY cipher, namely, the S-box, over an emulation of its Boolean circuit for MPC evaluation. Our approaches implement SKINNY's S-boxes over $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^8}$.

In the 4-bit case, we identified a favourable trade-off between the Boolean implementation, a direct interpolation of the S-box with squaring, and a polynomial decomposition approach. Choosing the decomposition approach saves 50% of multiplications in the circuit, traded-off with pre-processed random bits, compared to the Boolean implementation. Our practical benchmark confirms the trade-off. Moving to the arithmetic circuit setting indeed offers increased performance benefits of $\approx 18\%$ faster execution with $\approx 49\%$ less data.

In the 8-bit case, we observe that the S-box cannot be more efficiently expressed using our techniques. Our benchmark shows no improvement over the baseline Boolean circuit approach. Nonetheless, we find new parameters for the polynomial decomposition approach specific to SKINNY's 8-bit S-boxes that reduces the multiplicative depth of an evaluation from 9 to 5.

Further, we apply our technique to PHOTON and obtain an improved circuit representation with 50% fewer multiplications for the variants with 4-bit cells. For the 8-bit cell-based variant $P_{288}$ with the AES S-box optimization, we achieve a circuit with $\approx 81\%$ fewer multiplications. A practical benchmark confirms the optimization effort over a Boolean circuit emulation with 27% and 74% faster execution and 49% and 81% less data for $P_{100}$ and $P_{288}$, respectively.

Finally, we note that the identified polynomial decomposition approach will likely achieve similar improvements for other primitives with 4-bit S-boxes, such as Midori, TWINE, LED, KLEIN, QARMA, or KNOT.

# A Appendix

## A.1 Squaring is Linear

In the following, we briefly recall how squaring becomes a linear operation once the bit-decomposition of the finite field element is available. Let, for example, $b_3, b_2, b_1, b_0$ be the bits of $x \in \mathbb{F}_{2^4}$, then $x^2$ can be expressed as

$$
\begin{aligned}
& (b_3 X^3 + b_2 X^2 + b_1 X + b_0)^2 && \mod (X^4 + X^3 + 1) \\
={}& (b_3 X^3)^2 + (b_2 X^2)^2 + (b_1 X)^2 + b_0^2 && \mod (X^4 + X^3 + 1) \\
={}& b_3^2 X^6 + b_2^3 X^4 + b_1^2 X^2 + b_0^2 && \mod (X^4 + X^3 + 1) \\
={}& b_3 X^6 + b_2 X^4 + b_1 X^2 + b_0 && \mod (X^4 + X^3 + 1) \\
\equiv{}& (b_3 + b_2) X^3 + (b_3 + b_1) X^2 + b_3 X + (b_3 + b_2 + b_0)
\end{aligned}
$$

Note that $x^4$ follows naturally by setting $b_3' = b_3 + b_2$, $b_2' = b_3 + b_1$ etc. and repeating the computation from above.

## A.2 Additional Figures



(a) Shortest addition chain for powers in the interpolating polynomial for $\mathcal{S}_4$ using free squares $(2, 4, 8)$, $(6, 12, 9)$ and $(10)$ (SQ3). Note that since 6 is no longer available in round 2, 14 has to be computed in round 4.

(b) Trade-off between the number of multiplications and free squares for the interpolation polynomial of $\mathcal{S}_8$.
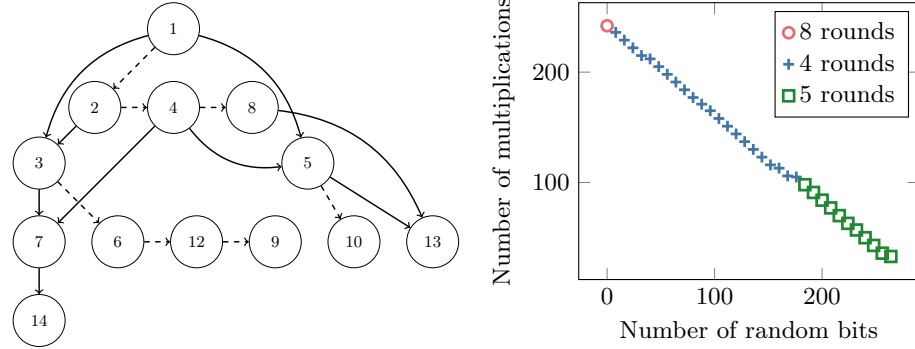
Fig. 5: Additional figures for shortest addition chain and the multiplication – free squares trade-off.

## A.3 Polynomial Decomposition Parameters

Table 7: Decomposition parameter for the forward and inverse 4-bit S-box of SKINNY and PHOTON. We give coefficient vectors for the polynomials for all powers in $L = (z^0, z^1, z^2, z^3, z^4, z^6, z^8, z^9, z^{12})$.

| S-box | Modulus | $p_i(z)$ |
|---|---|---|
| SKINNY forw. 4-bit | $X^4 + X^3 + 1$ | $q_1(z) = (0\text{x}0, 0\text{x}6, 0\text{x}4, 0\text{x}d, 0\text{x}3, 0\text{x}4, 0\text{x}8, 0\text{x}b, 0\text{x}8)$ <br> $p_1(z) = (0\text{x}1, 0\text{x}c, 0\text{x}f, 0\text{x}1, 0\text{x}5, 0\text{x}2, 0\text{x}b, 0\text{x}b, 0\text{x}0)$ <br> $p_2(z) = (0\text{x}c, 0\text{x}3, 0\text{x}0, 0\text{x}8, 0\text{x}e, 0\text{x}a, 0\text{x}0, 0\text{x}9, 0\text{x}0)$ |
| SKINNY inv. 4-bit | $X^4 + X^3 + 1$ | $q_1(z) = (0\text{x}0, 0\text{x}6, 0\text{x}4, 0\text{x}d, 0\text{x}3, 0\text{x}4, 0\text{x}8, 0\text{x}b, 0\text{x}8)$ <br> $p_1(z) = (0\text{x}7, 0\text{x}7, 0\text{x}7, 0\text{x}9, 0\text{x}3, 0\text{x}a, 0\text{x}c, 0\text{x}f, 0\text{x}7)$ <br> $p_2(z) = (0\text{x}3, 0\text{x}d, 0\text{x}4, 0\text{x}f, 0\text{x}5, 0\text{x}7, 0\text{x}0, 0\text{x}6, 0\text{x}0)$ |
| PHOTON 4-bit | $X^4 + X + 1$ | $q_1(z) = (0\text{x}9, 0\text{x}f, 0\text{x}6, 0\text{x}b, 0\text{x}7, 0\text{x}6, 0\text{x}c, 0\text{x}8, 0\text{x}a)$ <br> $p_1(z) = (0\text{x}a, 0\text{x}2, 0\text{x}0, 0\text{x}e, 0\text{x}1, 0\text{x}1, 0\text{x}4, 0\text{x}0, 0\text{x}3)$ <br> $p_2(z) = (0\text{x}9, 0\text{x}b, 0\text{x}5, 0\text{x}e, 0\text{x}4, 0\text{x}9, 0\text{x}5, 0\text{x}0, 0\text{x}0)$ |

## A.4 Embeddings

We detail the used (inverse) embeddings in Table 8. The inversion of the embedding of $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^8}$ only costs 4 and 8 random bits from $\mathcal{F}_{\mathsf{Bit}}$, respectively.

Table 8: The used embeddings from $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^8}$ into $\mathbb{F}_{2^{40}}$ on a bit level. Let $b_3 X^3 + b_2 X^2 + b_1 X + b_0$ be an element in $\mathbb{F}_{2^4}$ and $b_7 X^7 + b_6 X^6 + b_5 X^5 + b_4 X^4 + b_3 X^3 + b_2 X^2 + b_1 X + b_0$ be an element in $\mathbb{F}_{2^8}$. An element in $\mathbb{F}_{2^{40}}$ is $\sum_{i=0}^{39} b_i' Y^i$. Bits $b_i'$ that are not set below are 0.

| Embedding | $\mathbb{F}_{2^4}/\mathbb{F}_{2^8}$ to $\mathbb{F}_{2^{40}}$ | $\mathbb{F}_{2^{40}}$ to $\mathbb{F}_{2^4}/\mathbb{F}_{2^8}$ |
|---|---|---|
| $\mathbb{F}_{2^4} \hookrightarrow \mathbb{F}_{2^{40}}$ via $Y^{35} + Y^{20} + Y^5 + 1$ | $\begin{pmatrix} b_0' \\ b_5' \\ b_{10}' \\ b_{15}' \\ b_{20}' \\ b_{30}' \\ b_{35}' \end{pmatrix} = \begin{pmatrix} 1&1&1&0 \\ 0&1&1&0 \\ 0&0&0&1 \\ 0&0&1&0 \\ 0&1&0&0 \\ 0&0&0&1 \\ 0&1&0&0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$ | $\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 1&1&0&0 \\ 0&1&0&1 \\ 0&0&0&1 \\ 0&0&1&0 \end{pmatrix} \begin{pmatrix} b_0' \\ b_5' \\ b_{10}' \\ b_{15}' \end{pmatrix}$ |
| $\mathbb{F}_{2^8} \hookrightarrow \mathbb{F}_{2^{40}}$ via $Y^{35} + Y^{30} + Y^{25} + Y^{20} + Y^{10}$ | $\begin{pmatrix} b_0' \\ b_5' \\ b_{10}' \\ b_{15}' \\ b_{20}' \\ b_{25}' \\ b_{30}' \\ b_{35}' \end{pmatrix} = \begin{pmatrix} 1&0&1&0&0&1&1&1 \\ 0&1&1&1&1&0&0&0 \\ 0&1&0&0&0&0&1&0 \\ 0&0&0&1&0&0&1&0 \\ 0&1&0&1&0&1&0&0 \\ 0&1&1&1&1&0&1&0 \\ 0&1&0&0&1&0&0&1 \\ 0&1&1&0&0&0&0&0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix}$ | $\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1&1&1&0&1&0&1&0 \\ 0&1&1&0&0&1&0&0 \\ 0&1&1&0&0&1&0&1 \\ 0&1&0&1&0&1&0&0 \\ 0&0&0&1&0&1&0&1 \\ 0&0&1&1&1&0&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&1&1&1&0&0&1&1 \end{pmatrix} \begin{pmatrix} b_0' \\ b_5' \\ b_{10}' \\ b_{15}' \\ b_{20}' \\ b_{25}' \\ b_{30}' \\ b_{35}' \end{pmatrix}$ |

# References

1. Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part I. LNCS, vol. 10031, pp. 191–219 (2016). `https://doi.org/10.1007/978-3-662-53887-6_7`

2. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 430–454. Springer (2015). `https://doi.org/10.1007/978-3-662-46800-5_17`

3. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols. IACR Trans. Symmetric Cryptol. **2020**(3), 1–45 (2020). `https://doi.org/10.13154/tosc.v2020.i3.1-45`

4. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: ForkAE v.1 (2019)

5. Ankele, R., Banik, S., Chakraborti, A., List, E., Mendel, F., Sim, S.M., Wang, G.: Related-Key Impossible-Differential Attack on Reduced-Round Skinny. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10355, pp. 208–228. Springer (2017). `https://doi.org/10.1007/978-3-319-61204-1_11`

6. Beaver, D.: Efficient Multiparty Protocols Using Circuit Randomization. In: Feigenbaum, J. (ed.) Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Proceedings. Lecture Notes in Computer Science, vol. 576, pp. 420–432. Springer (1991). `https://doi.org/10.1007/3-540-46766-1_34`

7. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016). `https://doi.org/10.1007/978-3-662-53008-5_5`

8. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: SKINNY-AEAD and SKINNY-Hash. IACR Trans. Symmetric Cryptol. **2020**(S1), 88–131 (2020). `https://doi.org/10.13154/tosc.v2020.iS1.88-131`

9. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Proceedings. LNCS, vol. 4727, pp. 450–466. Springer (2007). `https://doi.org/10.1007/978-3-540-74735-2_31`

10. Brandão, L.T.A.N., Peralta, R.: NIST IR 8214C ipd NIST First Call for Multi-Party Threshold Schemes (Initial Public Draft) (2023). `https://doi.org/10.6028/NIST.IR.8214C.ipd`

11. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. J. Cryptol. **31**(3), 885–916 (2018). https://doi.org/10.1007/s00145-017-9273-9

12. Coron, J., Roy, A., Vivek, S.: Fast Evaluation of Polynomials over Binary Finite Fields and Application to Side-Channel Countermeasures. In: Batina, L., Robshaw, M. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop. LNCS, vol. 8731, pp. 170–187. Springer (2014). https://doi.org/10.1007/978-3-662-44709-3_10

13. Damgård, I., Keller, M.: Secure Multiparty AES . In: Sion, R. (ed.) Financial Cryptography and Data Security. pp. 367–374. Springer (2010). https://doi.org/10.1007/978-3-642-14577-3_31

14. Damgård, I., Keller, M., Larraia, E., Miles, C., Smart, N.P.: Implementing AES via an Actively/Covertly Secure Dishonest-Majority MPC Protocol. In: Visconti, I., Prisco, R.D. (eds.) Security and Cryptography for Networks - 8th International Conference, SCN 2012. Proceedings. Lecture Notes in Computer Science, vol. 7485, pp. 241–263. Springer (2012). https://doi.org/10.1007/978-3-642-32928-9_14

15. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security. Proceedings. Lecture Notes in Computer Science, vol. 8134, pp. 1–18. Springer (2013). https://doi.org/10.1007/978-3-642-40203-6_1

16. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty Computation from Somewhat Homomorphic Encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer (2012). https://doi.org/10.1007/978-3-642-32009-5_38

17. Damgård, I., Zakarias, R.W.: Fast Oblivious AES A Dedicated Application of the MiniMac Protocol. In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) Progress in Cryptology - AFRICACRYPT 2016 - 8th International Conference on Cryptology in Africa, Proceedings. Lecture Notes in Computer Science, vol. 9646, pp. 245–264. Springer (2016). https://doi.org/10.1007/978-3-319-31517-1_13

18. Delaune, S., Derbez, P., Vavrille, M.: Catching the Fastest Boomerangs Application to SKINNY. IACR Trans. Symmetric Cryptol. **2020**(4), 104–129 (2020). https://doi.org/10.46586/tosc.v2020.i4.104-129

19. Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: A Cipher with Low ANDdepth and Few ANDs per Bit. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 662–692. Springer (2018). https://doi.org/10.1007/978-3-319-96884-1_22

20. Dobraunig, C., Grassi, L., Guinet, A., Kuijsters, D.: Ciminion: Symmetric Encryption Based on Toffoli-Gates over Large Finite Fields. In: Canteaut, A., Standaert, F. (eds.) Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12697, pp. 3–34. Springer (2021). https://doi.org/10.1007/978-3-030-77886-6_1

21. Durak, F.B., Guajardo, J.: Improving the Efficiency of AES Protocols in Multi-Party Computation. In: Borisov, N., Díaz, C. (eds.) Financial Cryptography and

Data Security - 25th International Conference, FC 2021, Revised Selected Papers, Part I. Lecture Notes in Computer Science, vol. 12674, pp. 229–248. Springer (2021). `https://doi.org/10.1007/978-3-662-64322-8_11`

22. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In: Bailey, M., Greenstadt, R. (eds.) 30th USENIX Security Symposium, USENIX Security 2021. pp. 519–535. USENIX Association (2021)

23. Grassi, L., Lüftenegger, R., Rechberger, C., Rotaru, D., Schofnegger, M.: On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12106, pp. 674–704. Springer (2020). `https://doi.org/10.1007/978-3-030-45724-2_23`

24. Grassi, L., Onofri, S., Pedicini, M., Sozzi, L.: Invertible Quadratic Non-Linear Layers for MPC-/FHE-/ZK-Friendly Schemes over Fnp Application to Poseidon. IACR Trans. Symmetric Cryptol. **2022**(3), 20–72 (2022). `https://doi.org/10.46586/tosc.v2022.i3.20-72`

25. Grassi, L., Øygarden, M., Schofnegger, M., Walch, R.: From Farfalle to Megafono via Ciminion: The PRF Hydra for MPC Applications. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 14007, pp. 255–286. Springer (2023). `https://doi.org/10.1007/978-3-031-30634-1_9`

26. Grassi, L., Rechberger, C., Rotaru, D., Scholl, P., Smart, N.P.: MPC-Friendly Symmetric Key Primitives. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 430–443. ACM (2016). `https://doi.org/10.1145/2976749.2978332`

27. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: Rogaway, P. (ed.) Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Proceedings. Lecture Notes in Computer Science, vol. 6841, pp. 222–239. Springer (2011). `https://doi.org/10.1007/978-3-642-22792-9_13`

28. Hadipour, H., Bagheri, N., Song, L.: Improved Rectangle Attacks on SKINNY and CRAFT. IACR Trans. Symmetric Cryptol. **2021**(2), 140–198 (2021). `https://doi.org/10.46586/tosc.v2021.i2.140-198`

29. Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T.: Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms. IACR Trans. Symmetric Cryptol. **2020**(1), 43–120 (2020). `https://doi.org/10.13154/tosc.v2020.i1.43-120`

30. Keller, M.: MP-SPDZ: A Versatile Framework for Multi-Party Computation. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1575–1590. ACM (2020). `https://doi.org/10.1145/3372297.3417872`

31. Keller, M., Orsini, E., Rotaru, D., Scholl, P., Soria-Vazquez, E., Vivek, S.: Faster Secure Multi-party Computation of AES and DES Using Lookup Tables. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Proceedings. Lec-

ture Notes in Computer Science, vol. 10355, pp. 229–249. Springer (2017). `https://doi.org/10.1007/978-3-319-61204-1_12`

32. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 830–842. ACM (2016). `https://doi.org/10.1145/2976749.2978357`

33. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ Great Again. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10822, pp. 158–189. Springer (2018). `https://doi.org/10.1007/978-3-319-78372-7_6`

34. Liu, G., Ghosh, M., Song, L.: Security Analysis of SKINNY under Related-Tweakey Settings (Long Paper). IACR Trans. Symmetric Cryptol. **2017**(3), 37–72 (2017). `https://doi.org/10.13154/tosc.v2017.i3.37-72`

35. Lorünser, T., Wohner, F.: Performance Comparison of Two Generic MPC-frameworks with Symmetric Ciphers. In: Samarati, P., di Vimercati, S.D.C., Obaidat, M.S., Ben-Othman, J. (eds.) Proceedings of the 17th International Joint Conference on e-Business and Telecommunications, ICETE 2020 - Volume 2: SECRYPT. pp. 587–594. ScitePress (2020). `https://doi.org/10.5220/0009831705870594`

36. Mandal, K., Gong, G.: Can LWC and PEC be Friends?: Evaluating Lightweight Ciphers in Privacy-enhancing Cryptography. In: Fourth Lightweight Cryptography Workshop. NIST (2020)

37. Méaux, P., Journault, A., Standaert, F., Carlet, C.: Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. In: Fischlin, M., Coron, J. (eds.) Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9665, pp. 311–343. Springer (2016). `https://doi.org/10.1007/978-3-662-49890-3_13`

38. National Institute of Standards and Technology: Specification for the ADVANCED ENCRYPTION STANDARD (AES). Federal Information Processing Standards Publications 197 (2001). `https://doi.org/10.6028/NIST.FIPS.197-upd1`

39. National Institute of Standards and Technology: Secure Hash Standard (SHS). Federal Information Processing Standards Publications 180-4 (Aug 2015). `https://doi.org/10.6028/NIST.FIPS.202`

40. Sadeghi, S., Mohammadi, T., Bagheri, N.: Cryptanalysis of Reduced round SKINNY Block Cipher. IACR Trans. Symmetric Cryptol. **2018**(3), 124–162 (2018). `https://doi.org/10.13154/tosc.v2018.i3.124-162`

41. Zahur, S., Rosulek, M., Evans, D.: Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9057, pp. 220–250. Springer (2015). `https://doi.org/10.1007/978-3-662-46803-6_8`

42. Zhao, B., Dong, X., Meier, W., Jia, K., Wang, G.: Generalized related-key rectangle attacks on block ciphers with linear key schedule: applications to SKINNY and GIFT. Des. Codes Cryptogr. **88**(6), 1103–1126 (2020). `https://doi.org/10.1007/s10623-020-00730-1`