






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

Aysajan Abidin , Erik Pohle  , and Bart Preneel 

COSIC, KU Leuven, Leuven, Belgium

{aysajan.abidin,erik.pohle,bart.preneel}@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

This work is supported by the Flemish Government through FWO SBO project MOZAIK S003321N.

over correct hashing, ciphertext-compression for FHE schemes, and secure outsourcing of computation and data storage for MPC.

Applications of Symmetric Primitive Evaluation in MPC. The applications of MPC evaluation of symmetric-key primitives are numerous. We briefly sketch a selection of them. In a **distributed TPM**, instead of relying on trusted hardware, trust is distributed among multiple servers. Generation of secret keys is distributed and each server only ever obtains a secret share. The key shares are used in collaborative (or distributed) computations, e.g., encryption or signing, using MPC without reconstructing the secret key. Symmetric-key encryption can also be paired with MPC to enable flexible, secure and **privacy-preserving data collection and processing** [1]. Collected data can be encrypted at the source, stored and once MPC-based processing is desired, the data is decrypted in MPC and then processed. Since MPC creates a secure context for data processing, the input that is moved into this context and the output data that is moved out of this context may be encrypted to **facilitate secure input/output with parties that do not participate in the MPC protocol** [23,24]. Additionally, in the same way, MPC computation can be paused and continued later by encrypting intermediate data for secure storage. Finally, symmetric primitives in MPC may be used as **oblivious PRFs**, to bootstrap **secure database queries** or to create **MPC-in-the-head zero-knowledge proofs** and **post-quantum signatures** [10].

Related Work. Dedicated PRFs [18,23,24], block and stream ciphers [2-4,9,17,21,32], and hash functions [20,22] 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 [33] and SHA-2 [34], are widely used in real-world applications and are widespread in internet and industry standards. For instance, the correct processing of financial transaction data in MPC requires the usage of standardized constructions from that real-world domain since the information is not protected under non-standard cryptographic mechanisms that are MPC-friendly. 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 recently being investigated by NIST for standardization [8]. The important key part of thresholdization is that a threshold and a non-threshold implementation have to be interoperable, such that, e.g., systems managing keys in a threshold fashion can seamlessly interact with systems not using thresholdization.

While thresholdized AES implementations have been studied, e.g., [12,13,16,19,27], other traditional primitives have not received that much attention. In this work, we want to study threshold implementations of lightweight primi-

tives that may be used in applications where AES is undesirable. Lorünser and Wohner [30] 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. Motivated partly by the interoperability of privacy-enhancing protocols and lightweight cryptography, Mandal and Gong [31] study the Boolean circuit complexity of the core primitives in the NIST Lightweight Cryptography Competition (LWC)¹ round 2 candidates. However, their study is limited to Boolean circuits using the two-party garbling scheme HalfGates [35] for the MPC evaluation of the ciphers.

Contribution. 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, such as reduced circuit size, faster execution, or less 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 can replace bit-oriented functionality. For this purpose, we analyze the ten LWC finalists, but we limit our study to substitution-permutation network (SPN) designs of the underlying primitives which excludes SPARKLE, Grain-128AEAD, and TinyJambu. Moreover, we rule out the permutations used in sponge-based AEADs (Ascon, ISAP, and Xoodyak) for two reasons. First, 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. Further, 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 SKINNY serves as the main demonstration example, we also apply our findings to 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.

¹ <https://csrc.nist.gov/Projects/Lightweight-Cryptography>.

- 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 [6] is a lightweight tweakable block cipher with a SPN structure similar to AES. Its different variants process 64-bit or 128-bit blocks, and 64–384-bit tweakeys which is the concatenation of a (secret), e.g., 64- or 128-bit key and a (public) tweak. Table 1 lists the number of rounds specified for each variant. The round function alters the internal state, a 4×4 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 $s_0 s_1 \dots s_{15}$, then the 4×4 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 4×4 arrays, TK1, TK2, TK3, in the same manner. TK1, present in all variants, is loaded with tweak bits $0 \dots (16s - 1)$. TK2 and TK3 are loaded with tweak bits $16s \dots (32s - 1)$ and $32s \dots (48s - 1)$ respectively, if needed. The round function applies five steps in series: SubCells, AddRoundConstants, AddRoundKey, ShiftRows and MixColumns.

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 [6].

AddRoundConstants. This step XORs public constants to three cells:

$$s'_0 \leftarrow s_0 \oplus c_0, \quad s'_4 \leftarrow s_4 \oplus c_1, \quad s'_8 \leftarrow s_8 \oplus 0x2.$$

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

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 $a_i \oplus b_i \dots a_j \oplus b_j$, then

$$\begin{aligned} s'_{0..3} &\leftarrow s_{0..3} \oplus \text{TK1}_{0..3} \oplus \text{TK2}_{0..3} \oplus \text{TK3}_{0..3}, \\ s'_{4..7} &\leftarrow s'_{4..7} \oplus \text{TK1}_{4..7} \oplus \text{TK2}_{4..7} \oplus \text{TK3}_{4..7}. \end{aligned}$$

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

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

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 4×4 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 $\text{TK}i'$, $i = 1, 2, 3$, we have

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

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

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 [25]. 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 [7] 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 compute² 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 $\llbracket x \rrbracket_i = \langle x^{(i)}, m^{(i)} \rangle$ 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 $\mathcal{F}_{\text{Triple}}$ and \mathcal{F}_{Bit} by using somewhat homomorphic encryption SHE (e.g. in [14, 15, 29]) or oblivious transfer [28]. 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 $\mathcal{F}_{\text{Triple}}$ produces Beaver multiplication triples [5] of the form $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ where $c = ab$ and $a, b \xleftarrow{\$} \mathbb{F}_{2^k}$. The functionality \mathcal{F}_{Bit} produces random bits $\llbracket r \rrbracket$ 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 $\llbracket x \rrbracket$ 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, resulting in a so-called partial open. Before the final output is revealed, all MACs

² However, the players must know an upper bound on the number of required multiplication triples resp. random bits.

of partially opened shares are checked in one go. If this check passes, the output value is reconstructed.

Let $\llbracket x \rrbracket = \langle x^{(i)}, m_x^{(i)} \rangle$, $\llbracket y \rrbracket = \langle y^{(i)}, m_y^{(i)} \rangle$ 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:

$$\begin{aligned} e + \llbracket x \rrbracket &= \llbracket e + x \rrbracket : \begin{cases} \langle x^{(0)} + e, m_x^{(0)} + e\Delta^{(0)} \rangle & \text{if } i = 0, \\ \langle x^{(i)}, m_x^{(i)} + e\Delta^{(i)} \rangle & \text{else,} \end{cases} \\ e \cdot \llbracket x \rrbracket &= \llbracket e \cdot x \rrbracket : \langle e \cdot x^{(i)}, e \cdot m_x^{(i)} \rangle, \\ \llbracket x \rrbracket + \llbracket y \rrbracket &= \llbracket x + y \rrbracket : \langle x^{(i)} + y^{(i)}, m_x^{(i)} + m_y^{(i)} \rangle. \end{aligned}$$

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

1. The players partially open $\llbracket x - a \rrbracket$ as γ and $\llbracket y - b \rrbracket$ as ϵ .
2. Each player computes locally $\llbracket x \cdot y \rrbracket_i = \llbracket c \rrbracket_i + \gamma \cdot \llbracket b \rrbracket_i + \epsilon \cdot \llbracket a \rrbracket_i + \gamma \cdot \epsilon$.

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, \dots, 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 $\llbracket r_0 \rrbracket, \dots, \llbracket r_{k-1} \rrbracket$ from \mathcal{F}_{Bit} ,

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

In summary, multiplying two secret-shared values, i.e., $\llbracket x \cdot y \rrbracket \leftarrow \llbracket x \rrbracket \cdot \llbracket y \rrbracket$, requires one multiplication triple from $\mathcal{F}_{\text{Triple}}$ and one round of communication. A bit-decomposition of $\llbracket x \rrbracket$ into k bits $\llbracket b_0 \rrbracket, \dots, \llbracket b_{k-1} \rrbracket$ requires k random bits from \mathcal{F}_{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 fields³ of size 2^4 and 2^8 ,

$$\begin{aligned} \mathbb{F}_{2^4} &= \mathbb{F}_2[X]/(X^4 + X^3 + 1), \\ \mathbb{F}_{2^8} &= \mathbb{F}_2[X]/(X^8 + X^7 + X^6 + X^5 + X^4 + X^2 + 1). \end{aligned} \quad (1)$$

³ 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 $b_{s-1} \dots b_0$ into field elements as $b_{s-1} \dots b_0 \leftrightarrow \sum_{i=0}^{s-1} b_i X^i$. 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 Eq. (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.,

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

Given the bits of such a field element as vector $\mathbf{b} = (b_0, \dots, b_{s-1})$, the output bit vector for squaring is $\mathbf{sq} : \{0, 1\}^s \mapsto \{0, 1\}^s = \mathbf{M}\mathbf{b}$ where $\mathbf{M} \in \mathbb{F}_2^{s \times s}$ is a matrix depending on the irreducible polynomial. Thus, given the bit-decomposition \mathbf{b} of $x \in \mathbb{F}_{2^s}$, any power of the form x^{2^j} can be computed without any multiplication triples since \mathbf{sq} is a linear function. We stress, however, that the initial bit-decomposition requires one opening in the online phase, so computing any number of squares in $\{x^2, x^4, x^8, \dots\}$ costs one round of communication and s random bits.

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 [6] emulating Boolean operations (see Fig. 1). We will further use this approach as baseline for the comparison.

$$\begin{array}{lll}
 x'_0 \leftarrow x_1 \oplus (\neg x'_3 \wedge \neg x'_2) & x'_0 \leftarrow x_2 \oplus (\neg x'_3 \wedge \neg x'_1) & x'_4 \leftarrow x_3 \oplus (\neg x'_7 \wedge \neg x'_6) \\
 x'_1 \leftarrow x_2 \oplus (\neg x_1 \wedge \neg x'_3) & x'_1 \leftarrow x_7 \oplus (\neg x'_7 \wedge \neg x'_2) & x'_5 \leftarrow x_0 \oplus (\neg x_3 \wedge \neg x_2) \\
 x'_2 \leftarrow x_3 \oplus (\neg x_2 \wedge \neg x_1) & x'_2 \leftarrow x_6 \oplus (\neg x_2 \wedge \neg x_1) & x'_6 \leftarrow x_4 \oplus (\neg x_7 \wedge \neg x_6) \\
 x'_3 \leftarrow x_0 \oplus (\neg x_3 \wedge \neg x_2) & x'_3 \leftarrow x_1 \oplus (\neg x'_5 \wedge \neg x_3) & x'_7 \leftarrow x_5 \oplus (\neg x'_6 \wedge \neg x'_5)
 \end{array}$$

(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

$$\begin{pmatrix} 0x1 & 0x0^1 & \dots & 0x0^{2^s-1} \\ 0x1^0 & 0x1^1 & \dots & 0x1^{2^s-1} \\ & & \vdots & \\ & & & a_{2^s-1} \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_{2^s-1} \end{pmatrix} = \begin{pmatrix} \mathcal{S}_s(0x0) \\ \mathcal{S}_s(0x1) \\ \vdots \end{pmatrix}. \quad (3)$$

This approach primarily motivated the choice for the irreducible polynomials in Eq. (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 $a_i = 0x0$.

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

$$\begin{aligned}
 P_4(z) = & 0xc + 0x8z + 0x3z^2 + 0xdz^3 + 0xfz^4 + 0x4z^5 + 0x8z^6 + 0x6z^7 \\
 & + 0x1z^8 + 0x9z^9 + 0x8z^{10} + 0xez^{12} + 0xcz^{13} + 0xbz^{14}. \quad (4)
 \end{aligned}$$

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.

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,

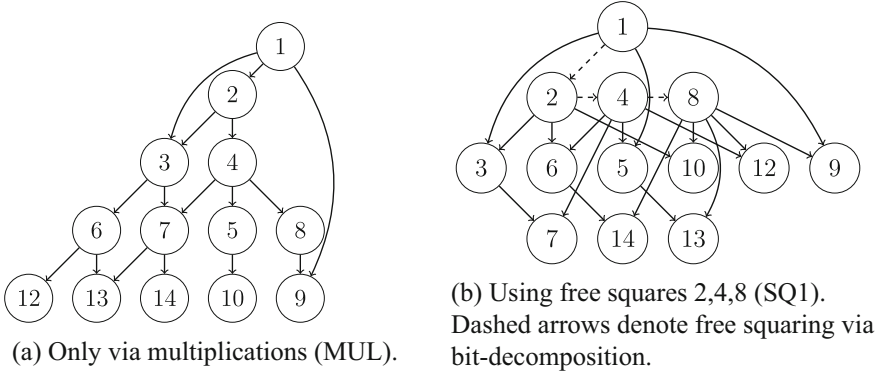


Fig. 2. Shortest addition chain for powers in the interpolating polynomial for SKINNY’s 4-bit S-box. Each level in the tree denotes one communication round.

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 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. 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 $1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow \dots$ 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.

3.4 Decomposition

We can use the decomposition method, CRV, by Coron, Roy and Vivek [11] 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 $p_i(z)$

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}			\mathcal{S}_8			\mathcal{S}_8^{-1}		
	Mult.	Bits	Depth	Mult.	Bits	Depth	Mult.	Bits	Depth	Mult.	Bits	Depth
MUL	12	0	4	11	0	4	242	0	8	239	0	8
BIN	4	0	2	4	0	4	8	0	4	8	0	4
SQ1	9	4	3	8	4	3	236	8	4	233	8	4
SQ2	6	8	3	6	8	3	33	264	5	32	256	5
SQ3	5	12	4	5	12	4	10	40	5	10	40	5
SQ4	3	16	4	3	16	4						
CRV	2	8	4	2	8	4						

and $q_i(z)$,

$$P(z) = \sum_{i=1}^{t-1} p_i(z)q_i(z) + p_t(z), \quad (5)$$

where each polynomial p_i, q_i only has monomials z^a with $a \in L$ where

$$L = C_{\alpha_1} \cup \dots \cup C_{\alpha_l}. \quad (6)$$

The set L is constructed from a number of cyclotomic bases C_{α_j} constructs the consecutive squares starting from α_j : $C_{\alpha_j} = \{2^i \alpha_j \pmod{2^s - 1} \mid \forall 0 \leq i < 2^s\}$.

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^{α_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 $p_i(z)q_i(z)$.

The CRV method is heuristic as one chooses the cyclotomic bases and coefficients for polynomials q_i to solve the resulting linear system for coefficients of p_i . The authors of [11] give α 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 [11]. 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.

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 Eq. (5).

	s	t	Base α	Mult	Bits	Depth
CRV [11]	4	2	{0, 1, 3}	2	8	4
CRV [11]	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

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-box⁴.

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 setting⁵ using the MASCOT MPC protocol [28] in the MP-SPDZ framework [26]. 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 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 Table 7 in Appendix A for more details). A different modulus for $\mathbb{F}_{2^{40}}$ would require different embeddings from \mathbb{F}_{2^4} and \mathbb{F}_{2^8} but has otherwise no impact on the performance.

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

⁴ The parameters cannot be used to decompose the AES S-box, for instance.

⁵ 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 [13]	1200	3200	45149	53

experiments more broadly usable, if, e.g., the key is the result of a previous MPC computation or each party inputs a key share as in the case for transcribing or OPRF evaluation.

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.

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.

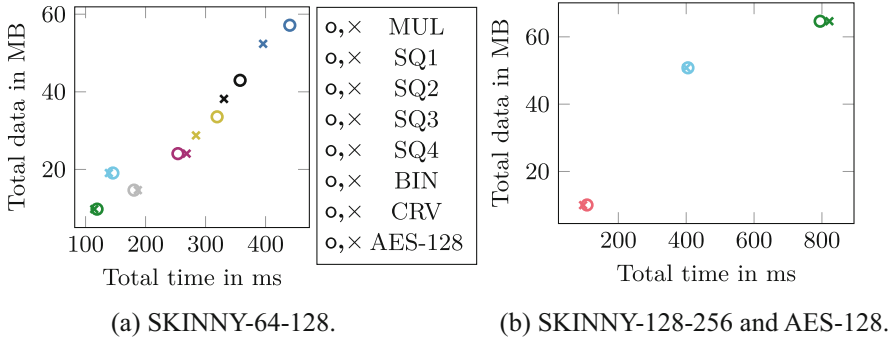


Fig. 3. Total, i.e., pre-processing and online, execution time and communication data for multiple S-box implementation approaches of SKINNY-64-128 and SKINNY-128-256 amortized with 100 executions in parallel. The legend symbol \circ denotes the forward direction while \times denotes the inverse direction.

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. [13].

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 [7] 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

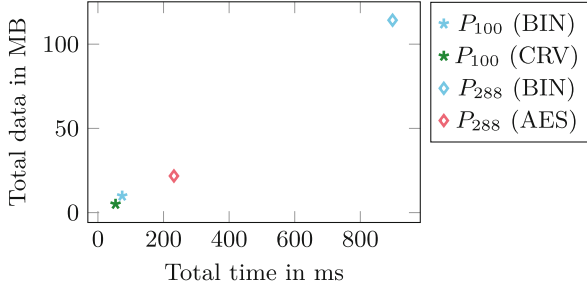


Fig. 4. Total, i.e., pre-processing and online, execution time and communication data for PHOTON P_{100} and P_{288} amortized with 100 executions in parallel.

known AES S-box optimizations from [13]. 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.

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.

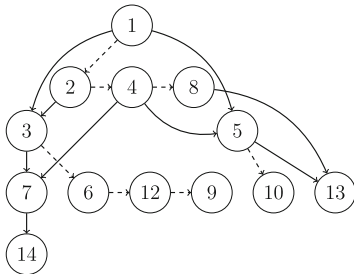
Finally, we note that the identified polynomial decomposition approach will likely achieve similar improvements for other primitives with 4-bit S-boxes, such as Midori, TWINE, LED, KLEIN, QARMA, or KNOT.

A Appendix

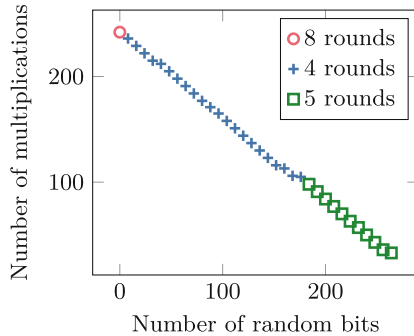
We detail the used (inverse) embeddings in Table 7. The inversion of the embedding of \mathbb{F}_{2^4} and \mathbb{F}_{2^8} only costs 4 and 8 random bits from \mathcal{F}_{Bit} , respectively.

Table 7. The used embeddings from \mathbb{F}_{2^4} and \mathbb{F}_{2^8} into $\mathbb{F}_{2^{40}}$ on a bit level. Let $b_3X^3 + b_2X^2 + b_1X + b_0$ be an element in \mathbb{F}_{2^4} and $b_7X^7 + b_6X^6 + b_5X^5 + b_4X^4 + b_3X^3 + b_2X^2 + b_1X + b_0$ be an element in \mathbb{F}_{2^8} . An element in $\mathbb{F}_{2^{40}}$ is $\sum_{i=0}^{39} b'_iY^i$. 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 $Y^{35} + Y^{20} + Y^{25}$ $+1$	$\begin{pmatrix} b'_0 \\ b'_5 \\ b'_{10} \\ b'_{15} \\ b'_{20} \\ b'_{30} \\ b'_{35} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$	$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} b'_0 \\ b'_5 \\ b'_{10} \\ b'_{15} \end{pmatrix}$
$\mathbb{F}_{2^8} \hookrightarrow \mathbb{F}_{2^{40}}$ via $Y^{35} + Y^{30} + Y^{25}$ $+Y^{20} + Y^{10}$	$\begin{pmatrix} b'_0 \\ b'_5 \\ b'_{10} \\ b'_{15} \\ b'_{20} \\ b'_{25} \\ b'_{30} \\ b'_{35} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix}$	$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} b'_0 \\ b'_5 \\ b'_{10} \\ b'_{15} \\ b'_{20} \\ b'_{25} \\ b'_{30} \\ b'_{35} \end{pmatrix}$



(a) Shortest addition chain for powers in the interpolating polynomial for 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 S_8 .

Fig. 5. Additional figures for shortest addition chain and the trade-off between multiplication and free squares.

References

1. Abidin, A., et al.: MOZAIK: an end-to-end secure data sharing platform. In: Second ACM Data Economy Workshop (DEC 2023), Seattle, WA, USA, 18 June 2023, p. 7. ACM (2023)
2. Albrecht, M., 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.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 191–219. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_7
3. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 430–454. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_17
4. 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)
5. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34
6. Beierle, C., et al.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 123–153. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_5
7. Bogdanov, A., et al.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_31
8. Brandão, L.T.A.N., Peralta, R.: NIST IR 8214C ipd NIST First Call for Multi-Party Threshold Schemes (Initial Public Draft) (2023)
9. Canteaut, A., et al.: Stream ciphers: a practical solution for efficient homomorphic-ciphertext compression. J. Cryptol. **31**(3), 885–916 (2018)
10. Chase, M., et al.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, pp. 1825–1842. Association for Computing Machinery (2017)
11. Coron, J.-S., Roy, A., Vivek, S.: Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 170–187. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_10
12. Damgård, I., Keller, M.: Secure multiparty AES. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 367–374. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14577-3_31
13. 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., De Prisco, R. (eds.) SCN 2012. LNCS, vol. 7485, pp. 241–263. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32928-9_14
14. 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.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40203-6_1

15. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38
16. Damgård, I., Zakarias, R.: Fast oblivious AES a dedicated application of the MiniMac protocol. In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2016. LNCS, vol. 9646, pp. 245–264. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31517-1_13
17. Dobraunig, C., et al.: Rasta: a cipher with low ANDdepth and few ANDs per bit. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 662–692. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_22
18. Dobraunig, C., Grassi, L., Guinet, A., Kuijsters, D.: CIMINION: symmetric encryption based on Toffoli-gates over large finite fields. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 3–34. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_1
19. Durak, F.B., Guajardo, J.: Improving the efficiency of AES protocols in multi-party computation. In: Borisov, N., Diaz, C. (eds.) FC 2021. LNCS, vol. 12674, pp. 229–248. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64322-8_11
20. 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)
21. 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.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 674–704. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_23
22. 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)
23. Grassi, L., Øyngarden, M., Schofnegger, M., Walch, R.: From Farfalle to Megafono via Ciminion: the PRF hydra for MPC applications. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023. LNCS, vol. 14007, pp. 255–286. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30634-1_9
24. 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)
25. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_13
26. Keller, M.: MP-SPDZ: a versatile framework for multi-party computation. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS 2020, pp. 1575–1590. ACM (2020)
27. 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.) ACNS 2017. LNCS, vol. 10355, pp. 229–249. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61204-1_12
28. 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)

29. Keller, M., Pastro, V., Rotaru, D.: Overdrive: making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 158–189. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_6
30. Lorünser, T., Wohnert, 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, SECRIPT, vol. 2, pp. 587–594. ScitePress (2020)
31. Mandal, K., Gong, G.: Can LWC and PEC be friends?: evaluating lightweight ciphers in privacy-enhancing cryptography. In: Fourth Lightweight Cryptography Workshop. NIST (2020)
32. Méaux, P., Journault, A., Standaert, F.-X., Carlet, C.: Towards stream ciphers for efficient FHE with low-noise ciphertexts. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 311–343. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_13
33. National Institute of Standards and Technology: Specification for the ADVANCED ENCRYPTION STANDARD (AES). Federal Information Processing Standards Publications 197 (2001)
34. National Institute of Standards and Technology: Secure Hash Standard (SHS). Federal Information Processing Standards Publications 180-4, August 2015
35. 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.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8