Faster Secure Multi-Party Computation of AES and DES Using Lookup Tables*

Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl,
Eduardo Soria-Vazquez, and Srinivas Vivek
{m.keller,emmanuela.orsini,dragos.rotaru,peter.scholl,
eduardo.soria-vazquez,sv.venkatesh}@bristol.ac.uk

Department of Computer Science, University of Bristol

Abstract. We present an actively secure protocol for secure multi-party computation based on lookup tables, by extending the recent, two-party 'TinyTable' protocol of Damgård et al. (ePrint 2016). Like TinyTable, an attractive feature of our protocol is a very fast and simple online evaluation phase. We also give a new method for efficiently implementing the preprocessing material required for the online phase using arithmetic circuits over characteristic two fields. This improves over the suggested method from TinyTable by at least a factor of 50.
As an application of our protocol, we consider secure computation of the Triple DES and the AES block ciphers, computing the S-boxes via lookup tables. Additionally, we adapt a technique for evaluating (Triple) DES based on a polynomial representation of its S-boxes that was recently proposed in the side-channel countermeasures community. We compare the above two approaches with an implementation. The table lookup method leads to a very fast online time of over 230,000 blocks per second for AES and 45,000 for Triple DES. The preprocessing cost is not much more than previous methods that have a much slower online time.

Keywords: multi-party computation, block cipher, implementation

1 Introduction

Secure multi-party computation (MPC) protocols allow useful computations to be performed on private data, without the data owners having to reveal their inputs. The last decade has seen an enormous amount of progress in the practicality of MPC, with many works designing more efficient protocols and implementations. There has also been a growing interest in exploring the possible applications of MPC, with a number of works targeting specific computations such as auctions, statistics and stable matching [BLW08,JKS08,BDJ+06,DES16].

* This work has been partially supported by EPSRC via grant EP/N021940/1; by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070; by EPSRC via grant EP/M016803; and by the European Union's H2020 Programme under grant agreement number ICT-644209 (HEAT) and the Marie Skodowska-Curie grant agreement No. 643161 (ECRYPT-NET).

One promising application area that has recently emerged is the use of secure computation to protect long-term secret keys, for instance, in authentication servers or to protect company secrets [AFL+16]. Here, the secret key, sk, is split up into n pieces, or shares, such that certain subsets of the n shares are needed to reconstruct sk, and each share is stored on a different server (possibly in a different location and/or managed by a separate entity). When the key is needed by an application, say for a user logging in, the servers run an MPC protocol to authenticate the user, without ever revealing sk. Typically, the type of computation required here can be performed using a symmetric primitive such as a block cipher or hash function.

Several previous works in secure computation have studied the above type of application, and the AES function is even considered a standard benchmark for new protocols [PSSW09,NNOB12,DKL+13,RR16,BLO16]. A recent line of works has even looked at special-purpose symmetric primitives, designed to have low complexity when evaluated in MPC [ARS+15,AGR+16,GRR+16]. However, in industries such as banking and the wider financial sector, strict regulations and legacy systems mean that switching to new primitives can be very expensive, or even impossible. Indeed, most banking systems today are using AES or Triple DES (3DES) to secure their data [EMV17], but may still benefit greatly from MPC technologies to prevent theft and data breaches.

1.1 Our Contributions

In this work, we focus on the task of secure multi-party computation of the AES and the (Triple) DES block ciphers, in the setting of active security against any number of corrupted parties. We present a new technique for the preprocessing phase of efficient, secure computation of *table lookup* (with a secret index), and apply this to evaluating the S-boxes of AES and DES. In addition, we describe a new method of secure MPC evaluation of the DES S-boxes based on evaluating polynomials over binary finite fields, which reduces the number of non-linear field multiplications.

Our protocol for secure table lookup builds upon the recent 'TinyTable' protocol for secure two-party computation by Damgård et al. [DNNR16]. This protocol requires a preprocessing phase, which is independent of the inputs, where randomly masked (or 'scrambled') lookup tables on random data are created. In the online phase, where the function is securely evaluated, each (one-time) masked table can be used to perform a single table lookup on a private index in the MPC protocol. The online phase of TinyTable is *very efficient*, as each party only needs to send $\log_2 N$ bits over the network, for a table of size N.

However, the suggested technique for creating the masked tables is far less efficient: for secure computation of AES, it would take at least *256 times longer* to create the masked lookup tables, compared with using standard methods with a slower online time.

We extend and improve upon the TinyTable approach in two ways. Firstly, we show that the technique can easily be generalized to the multi-party setting and

used in any SPDZ-like MPC protocol based on secret-sharing and information-theoretic MACs. Secondly, we describe a new, general approach for creating the masked tables using finite field arithmetic, which significantly improves the preprocessing cost of the protocol. Concretely, for a lookup table of size N, we can create the masked table using an arithmetic circuit over $\mathbb{F}_{2^k}$ with fewer than $N/k + \log N$ multiplications. This provides a range of possible instantiations with either binary or arithmetic circuit-based protocols. When using binary circuits, we only require $N - 2$ multiplications. For arithmetic circuits over $\mathbb{F}_{2^8}$, an AES S-box can be preprocessed with 33 multiplications, improving on the method in [DNNR16], which takes 1792 multiplications, by more than 50 times. With current practical protocols, it turns out to be even more efficient to work over $\mathbb{F}_{2^{40}}$, with only 11 multiplications. We remark that standard methods for computing AES based on polynomials or Boolean circuits can obtain better overall running times, but with a much slower online phase. The main goal of this work is to reduce the preprocessing cost whilst preserving the very fast online phase of TinyTable.

We also consider a new method for secure multi-party computation of DES based on a masking side-channel countermeasure technique. The DES S-box can be viewed as a lookup table mapping 6 bits to 4 bits, or as a polynomial over $\mathbb{F}_{2^6}$. A naïve method requires 62 field multiplications to evaluate a DES S-box polynomial over $\mathbb{F}_{2^6}$. There were many recent works that reduced the number of non-linear multiplications required to evaluate polynomials over binary finite fields, including the DES S-box polynomials [CGP+12,RV13,CRV14,CRV15,PV16]. A recent proposal by Pulkus and Vivek [PV16] showed that the DES S-boxes, when represented over a different field, $\mathbb{F}_{2^8}$, can be evaluated with only 3 non-linear multiplications. This is better than the best-known circuit over $\mathbb{F}_{2^6}$, which needs 4 non-linear multiplications. Applying the Pulkus–Vivek method in our context, we show how 1 round of the DES block cipher can be computed with just 24 multiplications over $\mathbb{F}_{2^8}$. This compares favorably with previous methods based on evaluating polynomials over $\mathbb{F}_{2^6}$ and boolean circuits.

Analogous to the MPC protocols based on table lookups, there are also masking side-channel countermeasures based on random-table lookups [CJRR99,Cor14]. This analogy should not come as a surprise since the masking technique is also based on secret-sharing. The state-of-the-art for (higher-order) masking seems to suggest that the schemes based on evaluation of S-box polynomials usually outperform table-lookups based schemes in terms of time, RAM memory and randomness. We perform a similar comparison in the MPC context too. To this end, we evaluate the complexity of the various methods for secure computation of AES and 3DES, and present some implementation results. We implemented the protocols using the online phase of the SPDZ [DPSZ12,DKL+13] MPC protocol. The preprocessing additionally requires some random multiplication triples and shared bits, for which we estimated costs using MASCOT [KOS16] for arithmetic circuits, and based on the recent optimized TinyOT protocol [NNOB12,WRK17] for binary circuits.

Our experiments show that the fastest online evaluation is achieved using lookup tables. The preprocessing for this method costs much less when using arithmetic circuits over larger fields, compared with a binary circuit protocol such as TinyOT [NNOB12,WRK17], despite the quadratic (in the field bit length) communication cost of [KOS16]. The polynomial-based methods for AES and DES still perform slightly better in the preprocessing phase, but for applications where a low online latency is desired, the lookup table approach is definitely preferred. If an application is mainly concerned with the total running time, then the polynomial-based methods actually lead to runtimes for AES that are comparable with the fastest recent 2-PC implementations using garbled circuits.

Related work. A recent, independent work by Dessouky et al. [DKS+17] presented two different protocols for lookup table-based secure two-party computation in the semi-honest security model. The first protocol, OP-LUT, offers an online phase very similar to ours (and [DNNR16]), while the preprocessing stage, that is implemented using 1-out-of-N oblivious transfer, is incomparable to ours as we must work much harder to achieve active security.
The second protocol, SP-LUT, proposes a more efficient preprocessing phase, which only requires random 1-out-of-N oblivious transfer computation, but a slower online evaluation; however this protocol has a much lower overall communication compared to the previous one. These two protocols are also compared with the OTTT (One-Time Truth-Table) protocol by Ishai et al. [IKM+13] with parallel circuit based preprocessing [DZ16]. More detailed comparisons with our protocols are provided in Section 5.2.

This work also provides an FPGA-based synthesis tool that transforms a high level function representation to multi-input/multi-output table-lookup representation, which could also be used with our protocol.

2 Preliminaries

We denote by λ the computational security parameter and κ the statistical security parameter. We consider the sets $\{0,1\}$ and $\mathbb{F}_2^k$ endowed with the structure of the fields $\mathbb{F}_2$ and $\mathbb{F}_{2^k}$, respectively. We denote by $\mathbb{F} = \mathbb{F}_{2^k}$ any finite field of characteristic two. Finally, we use $a \xleftarrow{\$} A$ as notation for a uniformly random sampling of a from a set A.

Note that by linearity we always mean $\mathbb{F}_2$-linearity, as we only consider fields of characteristic 2.

2.1 MPC Computation Model

Our protocol builds upon the *arithmetic black-box* model for MPC, represented by the functionality $\mathcal{F}_{\mathsf{ABB}}$ (Fig. 1) . This functionality permits the parties to input and output secret-shared values and evaluate arbitrary binary circuits performing basic operations. This abstracts away the underlying details of secret

Functionality $\mathcal{F}_{\mathsf{ABB}}$

Initialize: On input (Init, k) from all parties, store $\mathbb{F} = \mathbb{F}_{2^k} \cong \mathbb{F}_2[X]/f(X)$.

Input: On input $(\mathsf{Input}, P_i, \mathsf{id}, x)$ from P_i and $(\mathsf{Input}, P_i, \mathsf{id})$ from all other parties, with id a fresh identifier and $x \in \mathbb{F}$, store (id, x).

Add: On command $(\mathsf{Add}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$ from all parties (where $\mathsf{id}_1, \mathsf{id}_2$ are present in memory and id_3 is a fresh identifier), retrieve (id_1, x), (id_2, y) and store $(\mathsf{id}_3, x + y)$.

Mult: On command $(\mathsf{Mult}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$ from all parties (where $\mathsf{id}_1, \mathsf{id}_2$ are present in memory and id_3 is fresh identifier), retrieve (id_1, x), (id_2, y) and store $(\mathsf{id}_3, x \cdot y)$.

BitDec: On command $(\mathsf{BitDec}, \mathsf{id}, \mathsf{id}_0, \ldots, \mathsf{id}_{\ell-1})$ from all parties (where id is present in memory and $\mathsf{id}_0, \ldots, \mathsf{id}_{\ell-1}$ are fresh identifiers), retrieve (id, x) and store $(\mathsf{id}_i, x_i), i = 0, \ldots, \ell - 1$, where $x = \sum_{i=0}^{\ell-1} X^i \cdot x_i$.

Random: On input $(\mathsf{Random}, \mathsf{id})$ from all parties (where id is a fresh identifier), sample $x \xleftarrow{\$} \mathbb{F}$ and store (id, x).

RandomBit: On input $(\mathsf{RandomBit}, \mathsf{id}_b)$ from all parties (where id_b is a fresh identifier), sample $b \xleftarrow{\$} \mathbb{F}_2$ and store (id_b, b).

Open: On receiving $(\mathsf{Open}, \mathsf{id})$ from all parties, where $\mathsf{id} \in \mathsf{Val.keys}()$, send $\mathsf{Val}[\mathsf{id}]$, wait for x from the adversary, and output x to all parties.

Output: On command $(\mathsf{Output}, \mathsf{id})$ from all honest parties (where id is present in memory), retrieve (id, y) and output it to the adversary. Wait for an input from the adversary; if this is $\mathsf{Deliver}$ then output y to all parties, otherwise output Abort.

Fig. 1. The arithmetic black-box ideal functionality

sharing and MPC. Other than the standard **Add** and **Mult** commands, $\mathcal{F}_{\mathsf{ABB}}$ also has a **BitDec** command for generating the bit decomposition of a given secret-shared value, two commands **Random** and **RandomBit** for generating random values according to different distributions and an **Open** command which allows the parties and the adversary to output values. **BitDec** can be implemented in a standard manner by opening and then bit-decomposing $x + r$, where r is obtained using k secret random bits.

We use the notation $[\![x]\!]$ to denote an authenticated and secret-shared value x, which is stored by $\mathcal{F}_{\mathsf{ABB}}$. More precisely, this can be implemented with active security using the SPDZ protocol [DPSZ12,DKL+13] based on additive secret sharing and unconditionally secure MACs. We also use the $+$ and $\cdot$ operators to denote calls to **Add** and **Mul** with the appropriate shared values in $\mathcal{F}_{\mathsf{ABB}}$.

More concretely, our protocols are in the so called *preprocessing model* and consist of two different phases: an *online* computation, where the actual evaluation takes place, and a *preprocessing* phase that is independent of the parties' inputs. During the online evaluation, linear operations only require local computations thanks to the linearity of the secret sharing scheme and MAC. Multiplications and bit decompositions require random preprocessed data and interactions. More generally, the main task of the preprocessing step is to produce enough random secret data for the parties to use during the online computation: other than

multiplication triples, which allow parties to compute products, it also provides random shared values. The preprocessing phase can be efficiently implemented using OT-based protocols for binary circuits [BLN+15,FKOS15,WRK17] and arithmetic circuits [KOS16].

Security model. We describe our protocols in the universal composition (UC) framework of Canetti [Can01], and assume familiarity with this. Our protocols work with n parties from the set $\mathcal{P} = \{P_1, \ldots, P_n\}$, and we consider security against malicious, static adversaries, i.e. corruption may only take place before the protocols start, corrupting up to $n - 1$ parties.

3 Evaluating AES and DES S-box Polynomials

In this section, we recollect some of the previously known methods that aim to reduce the number of non-linear operations to evaluate univariate polynomials over binary finite fields, particularly, the AES and the DES S-boxes represented in this form. Note here that, by a non-linear multiplication, we mean those multiplications of polynomials that are neither multiplication by constants nor squaring operations. Since squaring is a linear operation in binary fields, once a monomial is computed, it can be repeatedly squared to generate as many more monomials as possible without costing any non-linear multiplication.

For the sake of completeness, we have included a description of DES and AES in Appendix A and B, respectively.

3.1 AES S-box

The AES S-box evaluation on a given input (as an element of $\mathbb{F}_{2^8}$) consists of first computing its multiplicative inverse in $\mathbb{F}_{2^8}$ (mapping zero to zero), and then applying a bijective affine transformation. For the inverse S-box, the inverse affine transformation is applied first and then the multiplicative inverse. Note that the polynomial representation of the inverse function in $\mathbb{F}_{2^8}$ is X^{254}.

BitDecompostion Method. This approach, described by Damgård et al. [DKL+12], computes the squares X^{2^i}, for $i \in [7]$, and then multiplies them to get X^{254}. This method needs 6 non-linear multiplications.

Rivain–Prouff Method. The method below is a variant, also used by Gentry et al. [GHS12], of the method of Rivain–Prouff [RP10] to evaluate the AES S-box polynomial using only 4 non-linear multiplications in $\mathbb{F}_{2^8}[X]$. The sequence of monomials computed is shown below:

$$\{X, X^2\} \xrightarrow{\times} \{X^3, X^{12}\} \xrightarrow{\times} \{X^{14}\} \xrightarrow{\times} \{X^{15}, X^{240}\} \xrightarrow{\times} X^{254}.$$

3.2 DES S-boxes

Cyclotomic Class Method. Recall that DES has eight 6-to-4-bit S-boxes. In this naïve method given by Carlet et al. [CGP+12], the DES S-boxes are represented as univariate polynomials over $\mathbb{F}_{2^6}$, where the S-box i/o bit strings are naturally identified with the elements of $\mathbb{F}_{2^6}$. In particular, the 4-bit S-box outputs are padded with zeros in the most significant bits and then identified with the elements of $\mathbb{F}_{2^6}$. By Lagrange interpolation, we know that these polynomials have degree at most 63. Indeed it turns out that they have degree at most 62 [RV13].

Over $\mathbb{F}_{2^m}[X]$, define

$$C_i^m := \left\{ X^{i \cdot 2^j} \ : \ j = 0, 1, \ldots, m-1 \right\} \text{ for } 0 < i < 2^m. \tag{1}$$

Note that $X^{2^m} = X$ in $\mathbb{F}_{2^m}[X]/(X^{2^m} + X)$. The cyclotomic class method is to first compute all the monomials up to degree 62 using as few non-linear multiplications as possible. As previously mentioned, once a monomial is computed, it is repeatedly squared to generate more monomials without costing additional non-linear multiplications. Once all the monomials are computed, the target polynomial is obtained simply as a linear combination of the computed monomials, hence without needing any further non-linear operations.

We need to compute 13 distinct classes of monomials to cover all monomials up to degree 62 in $\mathbb{F}_{2^6}[X]$:

$$C_0^6, C_1^6, C_3^6, C_5^6, C_7^6, C_9^6, C_{11}^6, C_{13}^6, C_{15}^6, C_{21}^6, C_{23}^6, C_{27}^6, C_{31}^6.$$

But since C_0^6 and C_1^6 can be computed without any non-linear multiplication from the input X, we have that each of the DES S-boxes can be computed with at most 11 non-linear multiplications, one for each of the remaining classes.

Pulkus–Vivek Method. This generic method to evaluate arbitrary polynomials over binary finite fields was proposed recently by Pulkus and Vivek [PV16] as an improvement over the method of Coron–Roy–Vivek [CRV14,CRV15]. In the PV method, the DES S-boxes are represented as polynomials over $\mathbb{F}_{2^8}$ instead of $\mathbb{F}_{2^6}$. The 6-bit input strings of the DES S-boxes are padded with zeroes in the two most significant positions and then naturally identified with the elements of $\mathbb{F}_{2^8}$. The four most significant coefficient bits of the polynomial outputs are *discarded* to obtain the desired 4-bit S-box output.

Firstly, a set of monomials $L = C_1^8 \cup C_3^8 \cup C_7^8$ in $\mathbb{F}_{2^8}[X]$ is computed. Recall from (1) that

$$\begin{aligned} C_1^8 &= \{X, X^2, X^4, X^8, X^{16}, X^{32}, X^{64}\}, \\ C_3^8 &= \{X^3, X^6, X^{12}, X^{24}, X^{48}, X^{96}, X^{65}\}, \\ C_7^8 &= \{X^7, X^{14}, X^{28}, X^{56}, X^{112}, X^{97}, X^{67}\}. \end{aligned}$$

Then a polynomial, say $P(X)$, representing the given S-box is sought in the following form

$$P(X) = p_1(X) \cdot q_1(X) + p_2(X),$$

where $p_1(X)$, $q_1(X)$, and $p_2(X)$ have monomials only from the set L. This decomposition, and the polynomial $P(X)$ itself, is obtained by setting up a linear system of equations over $\mathbb{F}_2$ in the bits of the unknown coefficients of $p_1(X)$ and $p_2(X)$. This is done by

1. randomly choosing the coefficients (corresponding only to the monomials in L) of q_1,
2. evaluating the above relation at every field element corresponding to the S-box inputs, and then obtaining one equation for every *bit* of the S-box output.

It has been observed in [PV16] that with high probability we obtain the above decomposition and in the process $P(X)$ itself. It is also possible to use a common $q_1(X)$ for all the eight DES S-boxes. In total, the PV method needs 3 non-linear multiplications in $\mathbb{F}_{2^8}[X]$ to evaluate each of the S-box polynomial: two to compute C_3^8 and C_7^8, and one to compute the product $p_1(X) \cdot q_1(X)$. Note that once L is computed, then $p_1(X)$, $q_1(X)$, and $p_2(X)$ are computed as linear combination of the monomials in L.

3.3 MPC Evaluation of AES and DES S-box Polynomials

Here we detail the MPC evaluation of AES and DES S-boxes using the techniques described above. We recall that since the S-boxes, in both the ciphers we are considering, are the only non-linear components, they represent the only parts which actually need interactions in an MPC evaluation.

AES Evaluation. As we mention before in Section 3.1, the straightforward way to compute the S-box is using the BitDecomposition method, which requires 6 multiplications in $4 + 1$ rounds. We are considering the case of active security, so the AES evaluation is done in the field $\mathbb{F}_{2^{40}}$ instead of $\mathbb{F}_{2^8}$, via the embedding $\mathbb{F}_{2^8} \hookrightarrow \mathbb{F}_{2^{40}}$. This follows from the fact that we are using the SPDZ protocol which requires a field size of at least 2^κ, where κ is the statistical security parameter. This permits to have only one MAC per data item [DKL+12].
The evaluation proceeds as follow: first X is bit-decomposed so that all the squarings can be locally evaluated, and then X^{254} is obtained as described in [DKL+12]:

$$X^{254} = ((X^2 \cdot X^4) \cdot (X^8 \cdot X^{16})) \cdot ((X^{32} \cdot X^{64}) \cdot X^{128}).$$

This requires 4 rounds, out of which one is a call to BitDec. We also need an extra round for computing the inverse of the field embedding $\mathbb{F}_{2^8} \hookrightarrow \mathbb{F}_{2^{40}}$ to evaluate the S-box linear layer. We denote this method by `AES-BD`.

We denote by `AES-RP` the AES S-box evaluation that uses the Rivain–Prouff method (cf. Section 3.1). It requires $6 + 1$ rounds to compute the four powers $X^3, X^{14}, X^{15}, X^{254}$. Furthermore, this can be done with three calls to BitDec and four non-linear multiplications, but some of the openings can be done in parallel which yields to a depth-6 circuit. As before, we need an extra round to call BitDec and compute the S-box linear layer.

Functionality $\mathcal{F}_{\mathsf{ABB-LUT}}$

This functionality has all the features of $\mathcal{F}_{\mathsf{ABB}}$, operating over $\mathbb{F}_{2^k}$, plus the following command.

Table Lookup: On command $(\mathsf{T}, \mathsf{id}_1, \mathsf{id}_2)$ from all parties, where $\mathsf{T} : \{0,1\}^\ell \to \{0,1\}^m$, for $\ell, m \leq k$, and id_1 is present in memory, retrieve (id_1, x) and store $(\mathsf{id}_2, \mathsf{T}(x))$.

Fig. 2. The ideal functionality for MPC using lookup tables

DES Evaluation. We denote by `DES-PV` the DES S-box evaluation using the Pulkus–Vivek method. Note that, although in side-channel world computing the squares is for free, since it is an $\mathbb{F}_2$-linear operation, in a secret-shared based MPC with MACs this is no longer true and we need to bit-decompose.

The squares from C_1^8, C_3^8, C_7^8, are obtained locally after X, X^3, X^7 are bit-decomposed. Here we need two multiplications, since $X^3 = X \cdot X^2$ and $X^7 = X^3 \cdot X^4$. The third multiplication occurs when computing the product $p_1(X) \cdot q_1(X)$, resulting in an S-box cost of only 3 triples, 24 bits and 5 communication rounds.

The number of rounds is due to 3 calls to BitDec (on X^3, X^7 and $p_1(X) \cdot q_1(X) + p_2(X)$) and 3 non-linear multiplications. Although at a first glance there seems to be six rounds, we have that $\mathsf{BitDec}(X^7)$ is independent of the $\mathsf{BitDec}(X^3)$, as we can compute X^7 without the call $\mathsf{BitDec}(X^3)$, resulting in only five rounds.

4 MPC Evaluation of Boolean Circuits using Lookup Tables

In this section we describe efficient MPC protocols for securely evaluating circuits over extension fields of $\mathbb{F}_2$ (including boolean circuits) containing additional 'lookup table' gates. The main protocol is in the preprocessing model and follows the same approach as the online phase in [DZ16], evaluating lookup table gates using preprocessed, masked lookup tables. In Section 4.1 we describe a variant of this method with circuit-dependent preprocessing, which leads to a faster online phase (similar to the TinyTable protocol [DNNR16]).

The functionality that we implement is $\mathcal{F}_{\mathsf{ABB-LUT}}$ (Figure 2), which augments the standard $\mathcal{F}_{\mathsf{ABB}}$ functionality with a table lookup command. The concrete online cost of each table lookup is just $\log_2 N$ bits of communication per party, where N is the size of the table. Note that the functionality $\mathcal{F}_{\mathsf{ABB-LUT}}$ works over a finite field $\mathbb{F}_{2^k}$, and has been simplified by assuming that the size of the range and domain of the lookup table T is not more than 2^k. However, our protocol actually works for general table sizes, and $\mathcal{F}_{\mathsf{ABB-LUT}}$ can easily be extended to model this by representing a table lookup result with several field elements instead of one.

Functionality $\mathcal{F}_{\mathsf{Prep-LUT}}$

This functionality has all of the same features as $\mathcal{F}_{\mathsf{ABB}}$, with the following additional command.

Masked Table: On input $(\mathsf{MaskedTable}, \mathsf{T}, \mathsf{id})$ from all parties, where $\mathsf{T} : \{0,1\}^\ell \to \{0,1\}^m$ for $\ell, m \leq k$, sample a random value s, set $(\mathsf{Val}[\mathsf{id}_s], \mathsf{Val}[\mathsf{id}_{\mathsf{T}(s)}], \ldots, \mathsf{Val}[\mathsf{id}_{\mathsf{T}(s\oplus(2^\ell-1))}]) \leftarrow (s, \mathsf{T}(s), \ldots, \mathsf{T}(s \oplus (2^\ell - 1)))$, and return $(\mathsf{id}_s, (\mathsf{id}_{\mathsf{T}(s)}, \ldots, \mathsf{id}_{\mathsf{T}(s\oplus(2^\ell-1))}))$.

Fig. 3. Ideal functionality for the preprocessing of masked lookup tables.

Protocol 1 Secure online evaluation of SBox using lookup tables

Table Lookup: On input $[\![x]\!]$ compute $[\![\mathsf{T}(x)]\!]$ as follows.

1. Call $\mathcal{F}_{\mathsf{Prep-LUT}}$ on input $(\mathsf{MaskedTable}, \mathsf{T})$, and obtain a precomputed masked table $([\![s]\!], [\![\mathsf{Table}(s)]\!])$.
2. The parties open the value $h = x \oplus s$.
3. Locally compute $[\![\mathsf{T}(x)]\!] = [\![\mathsf{Table}(s)]\!][h]$, where $[\![\mathsf{Table}(s)]\!][h]$ is the hth component of $[\![\mathsf{Table}(s)]\!]$.

We now show how Protocol 1 implements the **Table Lookup** command of $\mathcal{F}_{\mathsf{ABB-LUT}}$, given the right preprocessing material. For any non-linear function T, with ℓ input and m output bits, it is well known that it can be implemented as a lookup table of 2^ℓ components of m bits each. To evaluate $\mathsf{T}(\cdot)$ on a secret authenticated value $[\![x]\!], x \in \mathbb{F}_{2^\ell}$, the parties use a random authenticated T evaluation from $\mathcal{F}_{\mathsf{Prep-LUT}}$ (Figure 3). More precisely, we would like the preprocessing to output values $([\![s]\!], [\![\mathsf{Table}(s)]\!])$, where $[\![s]\!]$ is a random authenticated value unknown to the parties and $[\![\mathsf{Table}(s)]\!])$ is the table

$$[\![\mathsf{Table}(s)]\!] = \big([\![\mathsf{T}(s)]\!], [\![\mathsf{T}(s \oplus 1)]\!], \ldots, [\![\mathsf{T}(s \oplus (2^\ell - 1))]\!] \big),$$

so that $[\![\mathsf{Table}(s)]\!][j], 0 \leq j \leq 2^\ell - 1$, denotes the element $[\![\mathsf{T}(s \oplus j)]\!]$. Given such a table, evaluating $[\![\mathsf{T}(x)]\!]$ is straightforward: first the parties open the value $h = x \oplus s$ and then they locally retrieve the value $[\![\mathsf{Table}(s)]\!][h] = [\![\mathsf{T}(s \oplus h)]\!] = [\![\mathsf{T}(s \oplus s \oplus x)]\!] = [\![\mathsf{T}(x)]\!]$.

Correctness easily follows from the linearity of the $[\![\cdot]\!]$-representation and the discussion above. Privacy follows from the fact that the value s used in **Table Lookup** is randomly chosen and is used only once, thus it perfectly blinds the secret value x.

4.1 More Efficient Variant with TinyTable

The method just described is similar to, but not quite as efficient as, the approach in the two-party TinyTable protocol [DNNR16]. We can modify this to match the efficiency of TinyTable, generalized to the multi-party setting. The online cost of a secure lookup to a table $\mathsf{T} : \{0,1\}^\ell \to \{0,1\}^m$ becomes that of opening

an m-bit value, whereas the above method requires opening ℓ bits. This reduces the cost of an AND gate from opening 2 bits to just 1 bit, and the cost of a DES S-box from 6 bits to 4. Additionally, when implemented using SPDZ, the cost of linear operations becomes the same as computing the same operation on clear data; there is no need to also compute on (much larger) MACs, which gives a significant saving. The downside of this approach is that the preprocessing phase depends on the precise function being computed, so is less general.

We assume the gates in the circuit to be evaluated are all either $\mathbb{F}_2$-linear operations on bit vectors, or table lookup gates from $\{0,1\}^\ell \to \{0,1\}^m$. During the online phase, for each wire of the circuit, all parties will obtain a public value $x \oplus s$, where $x \in \{0,1\}^\ell$ is the *actual* value being computed on, and s is a random mask for that wire. The previous table lookup preprocessing is modified so that the i-th entry of the masked table contains a secret-sharing of $\mathsf{T}(s \oplus i) \oplus o$, where s and o are the random masks for the input and output wires (resp.) of that table lookup gate.

This means that $x \oplus s$ does not need to be opened in the online phase. Instead, the parties open the $(x \oplus s)$-th table entry, which is $[\![\mathsf{T}(x) \oplus o]\!]$. This gives them the public value for the output wire, which can be used in the next gate. Linear gates are computed in the clear on the public values. To obtain outputs at the end of the computation, the parties open the shared mask $[\![o]\!]$ (from the preprocessing) for every output wire. The online cost of each table lookup in this variant is that of opening an m bit value, instead of ℓ bits for the previous method. The preprocessing for the m-bit variant can be done with essentially the same cost as the previous section, but requires knowing the structure of the circuit in advance.

4.2 The Preprocessing Phase: Securely Generating Masked Lookup Tables

In this section we describe how to securely implement $\mathcal{F}_{\mathsf{Prep-LUT}}$ (see Figure 3), and in particular how to generate masked lookup tables which can be used for the online phase evaluation.

Recall that the goal is to obtain the shared values:

$$[\![\mathsf{Table}(s)]\!] = ([\![\mathsf{T}(s)]\!], [\![\mathsf{T}(s \oplus 1)]\!], \ldots, [\![\mathsf{T}(s \oplus (2^\ell - 1))]\!]).$$

Protocol 2 begins by taking a secret, random ℓ-bit mask $[\![s]\!] = ([\![s_0]\!], \ldots, [\![s_{\ell-1}]\!])$. Then, the parties expand s into a secret-shared bit vector $(s'_0, \ldots, s'_{2^\ell-1})$ which has a 1 in the s-th entry and is 0 elsewhere. We denote this procedure — the most expensive part of the protocol — by Demux, and describe how to perform it in the next section.

Once this is done, the parties can obtain the i-th entry of the masked lookup table by computing:

$$\mathsf{T}(i) \cdot [\![s'_0]\!] + \mathsf{T}(i \oplus 1) \cdot [\![s'_1]\!] + \cdots + \mathsf{T}(i \oplus (2^\ell - 1)) \cdot [\![s'_{2^\ell-1}]\!],$$

which is clearly $[\![\mathsf{T}(i \oplus s)]\!]$ as required. Note that since the S-box is public, this is a local computation for the parties. In the following we give an efficient protocol for computing Demux.

4.3 Computing Demux with Finite Field Multiplications

We now present a general method for computing Demux using fewer than $N/k + \log N$ multiplications over $\mathbb{F}_{2^k}$, when k is any power of 2 and $N = 2^\ell$ is the table size. Launchbury et al. [LDDA12] previously described a protocol with $O(N)$ multiplications in $\mathbb{F}_2$, but our protocol has fewer multiplications than theirs for all choices of k.

As said before, Demux maps a binary representation $(s_0, \ldots, s_{\ell-1})$ of an integer $s = \sum_{i=0}^{\ell-1} s_i \cdot 2^i$ into a unary representation of fixed length 2^ℓ that contains a one in the position s and zeros elsewhere. A straightforward way to compute Demux is by computing, over $\mathbb{F}_{2^N}$ [1]:

$$[\![s']\!] = \prod_{i=0}^{\ell-1} ([\![s_i]\!] \cdot X^{2^i} + (1 - [\![s_i]\!])).$$

Notice that if $s_i = 1$ then the i-th term of the product equals X^{2^i}, whereas the term equals 1 if $s_i = 0$. This means the entire product evaluates to $s' = X^s$, where s is the integer representation of the bits $(s_0, \ldots, s_{\ell-1})$. Bit decomposing s' obtains the demuxed output as required. Unfortunately, this approach does not scale well with N, the table size, as we must exponentially increase the size of the field.

We now show how to compute this more generally, using operations over $\mathbb{F}_{2^k}$, where k is a power of two. We will only ever perform multiplications between elements of $\mathbb{F}_2$ and $\mathbb{F}_{2^k}$, and will consider elements of $\mathbb{F}_{2^k}$ as vectors over $\mathbb{F}_2$. Define the partial products, for $j = 1, \ldots, \ell$:

$$p_j(X) = \prod_{i=0}^{j-1} (s_i \cdot X^{2^i} + (1 - s_i)) \in \mathbb{F}_{2^N}$$

[1] A similar trick was used by Aliasgari et al. [ABZS13] for binary to unary conversion over prime fields.

Protocol 2 Protocol to generate secret shared table lookup

Table: On input (Table, P_i) from all the parties, do the following:

1: Take ℓ random authenticated bits $[\![s_0]\!], \ldots, [\![s_{\ell-1}]\!]$, where each s_i is unknown to all the parties.
2: Compute $([\![s'_0]\!], \ldots, [\![s'_{2^\ell-1}]\!]) \leftarrow \mathsf{Demux}([\![s_0]\!], \ldots, [\![s_{\ell-1}]\!])$
3: $\forall i = 0, \ldots, 2^\ell - 1$, locally compute

$$[\![\mathsf{T}(i \oplus s)]\!] = \mathsf{T}(i) \cdot [\![s'_0]\!] + \mathsf{T}(i \oplus 1) \cdot [\![s'_1]\!] + \cdots + \mathsf{T}((2^\ell - 1) \oplus i) \cdot [\![s'_{2^\ell-1}]\!]$$

Protocol 3 $([\![s'_0]\!], \ldots, [\![s'_{N-1}]\!]) \leftarrow \mathsf{Demux}(k, [\![s_0]\!], \ldots, [\![s_{\ell-1}]\!])$

Require: k a power of two, $u = N/k$, $\ell = \log_2 N$
Input: Bit decomposition of $s \in \{0, \ldots, N-1\}$, with LSB first
Output: Satisfies $s'_s = 1$ and $s'_i = 0$ for all $i \neq s$

1: $[\![p]\!] = (1 - [\![s_0]\!], [\![s_0]\!])$ // p starts in $\mathbb{F}_2^2$
2: **for** $j = 1$ to $\ell - 1$ **do**
3: $\quad [\![t]\!] = [\![s_j]\!] \cdot [\![p]\!]$ // $\mathbb{F}_2 \times \mathbb{F}_2^{2^j}$ multiplication, 1 round
4: $\quad [\![p]\!] = (0^{2^j} \| [\![t]\!]) + ([\![p]\!] - [\![t]\!]) \| 0^{2^j})$ // p now in $\mathbb{F}_2^{2^{j+1}}$
5: Write $[\![p]\!] = ([\![b_0]\!], \ldots, [\![b_{u-1}]\!])$ // $b_i \in \mathbb{F}_2^k$
6: **for** $i = 0$ to $u - 1$ **do**
7: $\quad ([\![s'_{ki}]\!], \ldots, [\![s'_{ki+k-1}]\!]) = \mathsf{BitDec}([\![b_i]\!])$ // 1 round
8: **return** $([\![s'_0]\!], \ldots, [\![s'_{N-1}]\!])$

and note that $p_{j+1}(X) = p_j(X) \cdot (s_j \cdot X^{2^j} + (1 - s_j))$, for $j < \ell$.

Note also that the polynomial $p_j(X)$ has degree $< 2^j$, so $p_j(X)$ can be represented as a vector in $\mathbb{F}_2^{2^j}$ containing its coefficients, and more generally, a vector p_j containing $\lceil 2^j/k \rceil$ elements of $\mathbb{F}_2^k$. This is the main observation that allows us to emulate the computation of s' using only $\mathbb{F}_{2^k}$ arithmetic.

Given a sharing of p_j represented in this way, a sharing of $p_j(X) \cdot X^{2^j}$ can be seen as the vector (increasing the powers of X from left to right):

$$(0^{2^j} \| p_j) \in \mathbb{F}_2^{2^{j+1}}$$

and a vector representation of $p_{j+1}(X)$ is:

$$\left((0^{2^j} \| s_j \cdot p_j) + ((1 - s_j) \cdot p_j \| 0^{2^j})\right) \in \mathbb{F}_2^{2^{j+1}}.$$

Thus, given $[\![p_j]\!]$ represented as $\lceil 2^j/k \rceil$ shared elements of $\mathbb{F}_{2^k}$, we can compute $[\![p_{j+1}]\!]$ in MPC with $\lceil 2^j/k \rceil$ multiplications between $[\![s_j]\!]$ and a shared $\mathbb{F}_{2^k}$ element, plus some local additions.

Starting with $p_1(X) = s_0 \cdot X + (1 - s_0)$ we can iteratively apply the above method to compute $p_\ell = s'$, as shown in Protocol 3. The overall complexity of this protocol is given by

$$\sum_{j=1}^{\ell-1} \lceil 2^j/k \rceil < N/k + \ell$$

multiplications between bits and $\mathbb{F}_{2^k}$ elements.

Table 1 illustrates this trade-off between the field size and number of multiplications for some example parameters. We note that the main factor affecting the best choice of k is the cost of performing a multiplication in $\mathbb{F}_{2^k}$ in the underlying MPC protocol, and this may change as new protocols are developed. However, we compare costs of some current protocols in Section 5.

N	$k=1$	8	40	64	128
64	62	9	5	5	5
128	126	17	7	6	6
256	254	33	11	8	7
512	510	65	18	12	9
1024	1022	129	31	20	13

Table 1. Number of $\mathbb{F}_2 \times \mathbb{F}_{2^k}$ multiplications for creating a masked lookup table of size N, for varying k.

4.4 MPC Evaluation of AES and DES using Lookup Tables

We now show how to use the lookup table protocol from the previous section to evaluate AES and DES in MPC. We use the more general method from Protocol 1, and leave an implementation of the faster variant with circuit-dependent preprocessing to future work.

AES Evaluation. We require an MPC protocol which performs operations in $\mathbb{F}_{2^8}$. In practice, we actually embed $\mathbb{F}_{2^8}$ in $\mathbb{F}_{2^{40}}$, since we use the SPDZ protocol which requires a field size of at least 2^κ, for statistical security parameter κ. We implement the AES S-box using the table lookup method from Protocol 2 combined with Demux (Protocol 3) over $\mathbb{F}_{2^{40}}$, since this yields a lower communication cost (see Table 5). Notice that the data sent is highly dependent on the number of bits, triples and the field size.

In a naive implementation of this approach, we would have call BitDec on $[\![\mathsf{Table}(s)]\!]$, in order to perform the embedding $\mathbb{F}_{2^8} \hookrightarrow \mathbb{F}_{2^{40}}$. This is required since the table output is not embedded, but the `MixColumns` step needs this to perform multiplication by $X \in \mathbb{F}_{2^8}$ on each state.

With a more careful analysis we can avoid the BitDec calls by locally embedding the bit shares inside Protocol 2. We store the masked S-box table in bit decomposed form and then its bits are multiplied (in the clear) with Demux's output (secret-shared). This trick reduces the online communication by a factor of 8, halves the number of rounds required to evaluate AES and gives a very efficient online phase with only 10 rounds and 160 openings in $\mathbb{F}_{2^{40}}$.

DES Evaluation. Using the fact that DES S-boxes have size 64, we chose to use the Demux protocol 3 with multiplications in $\mathbb{F}_{2^{40}}$, based on the costs in Table 5. Like AES, we try to isolate the input-dependent phase as much as possible with no extra cost.

Every DES round performs only bitwise addition and no embedding is necessary here. The masked table can be bit-decomposed without interaction, exactly as described above for AES, by multiplying clear bits with secret shared values. This yields a low number of openings, one per S-box look-up, so the total online cost for `3DES` is 46 rounds with 384 openings.

5 Performance Evaluation

This section presents timings for `3DES` and `AES` using the methods presented in previous sections. We also discuss trade-offs and different optimizations which turn out to be crucial for our running-times. The setup we have considered is that both the key and message used in the cipher are secret shared across two parties. We consider the input format for each block cipher as already embedded into $\mathbb{F}_{2^{40}}$ for AES, or as a list of shared bits for DES. We implemented the protocols using the SPDZ software,[2] and estimated times for computing the multiplication triples and random bits needed based on the costs of MASCOT [KOS16].

The results, shown in Tables 2, 3 and 4, give measurements in terms of *latency* and *throughput*. Latency indicates the online phase time required to evaluate one block cipher, whereas throughput (which we consider for both online and offline phases) shows the maximum number of blocks per second which can be evaluated in parallel during one execution. We also measure the number of rounds of interaction of the protocols, and the number of *openings*, which is the total number of secret-shared field elements opened during the online evaluation.

Benchmarking Environment. The experiments were ran across two machines each with Intel i7-4790 CPUs running at 3.60GHz, 16GB of RAM connected over a 1Gbps LAN with an average ping of 0.3ms (round-trip). The WAN experiments were simulated using the Linux `tc` tool with an average ping latency of 100ms (round-trip) and a bandwidth of 50Mbps.

For experiments with 3–5 parties, we used three additional machines with i7-3770 CPUs at 3.1GHz. In order to get accurate timings each experiment was averaged over 5 executions, each with at least 1000 cipher calls.

Security Parameters and Field Sizes. Secret-sharing based MPC can be usually split into 2 phases — preprocessing and online. In SPDZ-like systems, the preprocessing phase depends on a computational security parameter, and the online phase a statistical security parameter which depends on the field size. In our experiments the computational security parameter is $\lambda = 128$. The statistical security κ is 40 for every cipher except for `3DES-Raw` which requires an embedding into a 42 bit field.

Results. The theoretical costs and practical results are shown in Table 2 and Table 3, respectively. Timings are taken only for the encryption calls, excluding the key schedule mechanism.

`AES-BD` is implemented by embedding each block into $\mathbb{F}_{2^{40}}$, and then squaring the shares locally after the inputs are bit-decomposed. In this manner, each S-box computation costs 5 communication rounds and 6 multiplications. This method was described in [DKL+12].

`3DES-Raw` represents the `3DES` cipher with the S-box evaluated as a polynomial of degree 62 over the field $\mathbb{F}_{2^6} = \mathbb{F}_2[x]/(x^6 + x^4 + x^3 + x + 1)$. To make the comparisons relevant with other ciphers in terms of active security we chose to embed the S-box input in $\mathbb{F}_{2^{42}}$, via the embedding $\mathbb{F}_{2^6} \hookrightarrow \mathbb{F}_{2^{42}}$,

[2] https://github.com/bristolcrypto/SPDZ-2

where $\mathbb{F}_{2^{42}} = \mathbb{F}_2[y]/(y^{42} + y^{21} + 1)$ and $y = x^7 + 1$. The S-boxes used for interpolating are taken from the PyCrypto library [Lit17]. `3DES-Raw` is implemented only for benchmarking purposes and it has no added optimizations. One S-box has a cost of 62 multiplications and 62 rounds.

`3DES-PV` is `3DES` implemented with the Pulkus-Vivek method from Section 3.2. Since it has only a few multiplications in $\mathbb{F}_{2^{40}}$, the amount of preprocessing data required is very small, close to `AES-BD`. It suffers in terms of both latency and throughput due to the high number of communication rounds (needed for bit decomposition to perform the squarings).

Surprisingly, `AES-RP` (the polynomial-based method from Section 3.1) has a better throughput than `AES-BD` although it requires 20 more rounds and 2 times more shared bits to evaluate. The explanation for this is that in `AES-RP` there are fewer openings, thus less data sent between parties. However, for the WAN experiments in `AES-RP` the latency increases dramatically because of the extra rounds and the round-trip time.

`AES-LT` and `3DES-LT` are the ciphers obtained with the lookup table protocol from Section 4. `AES-LT` achieves the lowest latency and the highest throughput in the online phase for both LAN and WAN settings. The communication in the preprocessing phase is roughly twice the cost of the previous method, `AES-BD`.

Packing optimization. We notice that in the online phase of `AES-LT` each opening requires to send 8 bit values embedded in $\mathbb{F}_{2^{40}}$. Instead of sending 40 bits across the network we select only the relevant bits, which for `AES-LT` are 8 bits. This reduces the communication by a factor of 5 and gives a throughput of 236k AES/second over LAN and a multi-threaded MPC engine.

The same packing technique is applied for `3DES-LT` since during the protocol we only open 6 bit values from Protocol 1. These bits are packed into a byte and sent to the other party. Here the multi-threaded version of `3DES-LT` improves the throughput only by a factor of 4.2x (vs `AES-LT` 4.5x) due to the higher number of rounds and openings along with the loss of 2 bits from packing.

Computation vs. Communication. Notice that for AES the throughput in the WAN setting compared to LAN decreases by at least 8 times. Surprisingly, the throughput of 3DES decreases by at most four - single threaded `3DES-LT` can perform around 10000 ops/s over LAN whereas over WAN it has 300 ops/s, same ratio preserves for the multi-threaded variant. Profiling suggests that AES has a lower computation cost than 3DES. This means that increasing the round-trip time between machines has a slightly worse effect for AES than 3DES since the CPU can do more work between subsequent rounds.

General costs of the table lookup protocol. In Table 5, we estimate the communication cost for creating preprocessed, masked tables for a range of table sizes, using our protocol from Section 4.2. This requires multiplication triples

over $\mathbb{F}_{2^k}$, where k is a parameter of the protocol. When $k = 1$, we give figures using a recent optimized variant [WRK17] of the two-party TinyOT protocol [NNOB12]. For larger choices of k, the costs are based on the MASCOT protocol [KOS16]. We note that even though MASCOT has a communication complexity in $O(k^2)$, it still gives the lowest costs (with $k = 40$) for all the table sizes we considered.

Cipher	Online cost			Preprocessing cost			
	Rounds	Openings	Field	Triples	Bits	Field	Comm.(MB)
AES-BD	50	2240	$\mathbb{F}_{2^{40}}$	960	2560	$\mathbb{F}_{2^{40}}$	4.3
AES-RP	70	1920	$\mathbb{F}_{2^{40}}$	640	5120	$\mathbb{F}_{2^{40}}$	2.9
AES-LT	10	160	$\mathbb{F}_{2^{40}}$	1760	42240	$\mathbb{F}_{2^{40}}$	8.4
3DES-Raw	2979	48024	$\mathbb{F}_{2^{42}}$	23808	2688	$\mathbb{F}_{2^{42}}$	112
3DES-PV	230	3456	$\mathbb{F}_{2^{40}}$	1152	9216	$\mathbb{F}_{2^{40}}$	5.2
3DES-LT	46	384	$\mathbb{F}_{2^{40}}$	1920	26880	$\mathbb{F}_{2^{40}}$	8.8

Table 2. Communication cost for AES and 3DES in MPC.

Cipher	Online (single-thread)			Online (multi-thread)			Preprocessing[a]
	Latency (ms)	Batch size	ops/s	Batch size	ops/s	Threads	ops/s
AES-BD	5.20	64	758	1024	3164	16	30.7
AES-RP	7.19	1024	940	64	3872	16	**46.1**
AES-LT	**0.928**	1024	51654	512	**236191**	32	16.79
3DES-Raw	270	512	130	-	-	-	1.24
3DES-PV	36.98	512	86	512	366	32	**25.6**
3DES-LT	**4.254**	1024	10883	512	**45869**	16	15.3

Table 3. 1 Gbps LAN timings for evaluating AES and 3DES in MPC.

[a] Extrapolated from timings for a 128-bit field

5.1 Multiparty Setting.

We also ran the AES-LT protocol with different numbers of parties and measured the throughput of the preprocessing and online phases. Figure 4 indicates that the preprocessing gets more expensive as the number of parties increases, whereas the online phase throughput does not decrease by much. This is likely

Cipher	Online (single-thread)			Online (multi-thread)			Preprocessing
	Latency (ms)	Batch size	ops/s	Batch size	ops/s	Threads	ops/s
`AES-BD`	2550	4096	79	2048	325	32	1.52
`AES-RP`	3569	4096	83	4096	346	32	**2.28**
`AES-LT`	**510**	4096	928	4096	**29055**	32	0.83
`3DES-PV`	11727	2048	35	512	185	32	**1.27**
`3DES-LT`	**2344**	4096	383	4096	**12165**	32	0.76

Table 4. 50 Mbps WAN timings for evaluating `AES` and `3DES` in MPC.

N	$k = 1$	40	64	128
64	35.01	21.8	43.52	112.64
128	71.16	30.52	52.22	135.17
256	143.45	47.96	69.63	157.7
512	288.02	78.48	104.45	202.75
1024	577.17	135.16	174.08	292.86

Table 5. Total communication cost (kBytes) of the $\mathbb{F}_2 \times \mathbb{F}_{2^k}$ multiplications needed in creating a masked lookup table of size N, with two parties. The $k = 1$ estimates are based on TinyOT [WRK17], the others on MASCOT [KOS16].

to be because the bottleneck for the preprocessing is in terms of communication (which is $O(n^2)$ in total), whereas the online phase is more limited by the local computation done by each party.

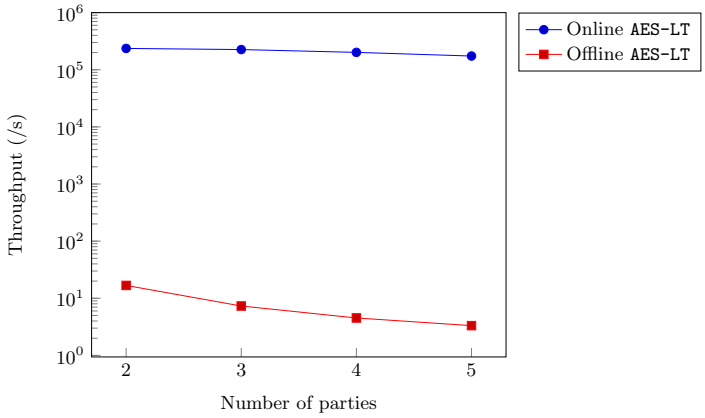

Fig. 4. Table lookup-based AES throughput for multiple parties.

5.2 Comparison with Other Works

We now compare the performance of our protocols with other implementations in similar settings. Table 6 gives an overview of the most relevant previous works. We see that our `AES-LT` protocol comes very close to the best online throughput of TinyTable, whilst having a far more competitive offline cost. [3] Our `AES-RP` variant has a slower online phase, but is comparable to the best garbled circuit protocols overall.

Protocol	Online		Comms.	Notes
	Latency (ms)	Throughput (/s)	(total)	
TinyTable (binary) [DNNR16]	4.18	24500	3.07 MB	
TinyTable (optim.) [DNNR16]	1.02	339000	786.4 MB	
Wang et al. [WRK17]	0.93	1075	2.57 MB	10 Gbps
Rindal-Rosulek [RR16]	1.0	1000	1.6 MB	10 Gbps
OP-LUT [DKS+17]	5	41670	0.103 MB	passive
SP-LUT [DKS+17]	6	2208	0.044 MB	passive
`AES-LT`	0.93	236200	8.4 MB	
`AES-RP`	7.19	940	2.9 MB	

Table 6. Performance comparison with other 2-PC protocols for evaluating AES in a LAN setting.

TinyTable Protocol. The original, 2-party TinyTable protocol [DNNR16] presented implementations of the online phase only, with two different variants. The fastest variant is based on table lookup and obtains a throughput of around 340 thousand AES blocks per second over a 1Gbps LAN, which is 1.51x faster than our online throughput. The latency (for sequential operations) is around 1ms, the same as ours. We attribute the difference in throughput to the additional local computation in our implementation, since we need to compute on MACs for every linear operation (this could be avoided if we used the protocol from Section 4.1.

TinyTable does not report figures for the preprocessing phase. However, we estimate that using TinyOT and the naive method suggested in the paper would need would need over 1.3 million TinyOT triples for AES (34 ANDs for each S-box, repeated 256 times to create one masked table, for 16 S-boxes in 10 rounds). In contrast, our table lookup method uses around 160 thousand TinyOT triples, or just 2080 triples over $\mathbb{F}_{2^{40}}$ (cf. Table 1), per AES block.

[3] The reason for the very large preprocessing cost of TinyTable is due to the need to evaluate the S-box 256 times per table lookup.

Garbled Circuits. There are many implementations of AES for actively secure 2-PC using garbled circuits [LR15,RR16,NST17,WMK17,WRK17]. When measuring online throughput in a LAN setting, using garbled circuits gives much worse performance than methods based on table lookup, because evaluating a garbled circuit is much more expensive computationally. For example, out of all these works the lowest reported online time (even over a 10Gbps LAN) is 0.93ms [WRK17], and this does not improve in the amortized setting.

Some recent garbled circuit implementations, however, improve upon our performance in the preprocessing phase, where communication is typically the bottleneck. Wang et al. [WRK17] require 2.57MB of communication when 1024 circuits are being garbled at once, while Rindal and Rosulek [RR16] need only 1.6MB. The runtime for both of these preprocessing phases is around 5ms over a 10Gbps LAN; this would likely increase to at least 15–20ms in a 1Gbps network, whereas our table lookup preprocessing takes around 60ms using MASCOT. If a very fast online time is not required, our implementation of the Rivain–Prouff method would be more competitive, since this has a total amortized time of only 23ms per AES block.

Secret-Sharing Based MPC. Other actively implementations of AES/DES using secret-sharing and dishonest majority based on secret sharing include those using SPDZ [DKL+12,KSS13] and MiniMAC [DZ13,DLT14]. Our `AES-BD` method is the same as [DKL+12] and obtains faster performance than both SPDZ implementations. For DES, our TinyTable approach improves upon the times of the binary circuit implementation from [KSS13] (which are for single-DES, so must be multiplied by 3) by over 100 times. Regarding MiniMAC, the implementation of [DLT14] obtains slower online phase times than our work and TinyTable, and it is not known how to do the preprocessing with concrete efficiency.

OP-LUT and SP-LUT . The proposed 2-party protocols by Dessouky et al. [DKS+17] only offer security in the semi-honest setting. The preprocessing phase for both the protocols are based on 1-out-of-N oblivious transfer. In particular, the cost of the OP-LUT setup is essentially that of 1-out-of-N OT, while the cost of SP-LUT is the cost of 1-out-of-N *random* OT, which is much more efficient in terms of communication.

The online communication cost of OP-LUT is essentially the same as our online phase, since both protocols require each party to send $\log_2 N$ bits for a table of size N. However, we incur some additional local computation costs and a MAC check (at the end of the function evaluation) to achieve active security. The online phase of SP-LUT is less efficient, but the overall communication of this protocol is very low, only 0.055MB for a single AES evaluation over a LAN setting with 1GB network.

The work [DKS+17] reports figures for both preprocessing and online phase: using OP-LUT gives a latency of around 5ms for 1 AES block in the LAN setting, and a throughput of 42000 blocks/s. These are both slower than our

online phase figures using `AES-LT`. The preprocessing runtimes of both OP-LUT and SP-LUT are much better than ours, however, achieving over 1000 blocks per second (roughly 80 times faster than `AES-LT`). This shows that we require a large overhead to obtain active security in the preprocessing, but the online phase cost is the same, or better.

Acknowledgements

We are grateful to Carsten Baum, Nigel Smart and the anonymous reviewers for valuable feedback that helped to improve the paper.

References

ABZS13. Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. Secure computation on floating point numbers. In *NDSS 2013*. The Internet Society, February 2013.

AFL+16. Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 805–817. ACM Press, October 2016.

AGR+16. Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219. Springer, Heidelberg, December 2016.

ARS+15. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, April 2015.

BDJ+06. Peter Bogetoft, Ivan Damgård, Thomas Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft. A practical implementation of secure auctions based on multiparty integer computation. In Giovanni Di Crescenzo and Avi Rubin, editors, *FC 2006*, volume 4107 of *LNCS*, pages 142–147. Springer, Heidelberg, February / March 2006.

BLN+15. Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High performance multi-party computation for binary circuits based on oblivious transfer. Cryptology ePrint Archive, Report 2015/472, 2015. http://eprint.iacr.org/2015/472.

BLO16. Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 578–590. ACM Press, October 2016.

BLW08. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS 2008*, volume 5283 of *LNCS*, pages 192–206. Springer, Heidelberg, October 2008.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CGP+12. Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for S-Boxes. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 366–384. Springer, 2012.

CJRR99. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO 1999, Proc.*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.

Cor14. Jean-Sébastien Coron. Higher order masking of look-up tables. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014. Proc.*, volume 8441 of *LNCS*, pages 441–458. Springer, 2014.

CRV14. Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014. Proc.*, volume 8731 of *LNCS*, pages 170–187. Springer, 2014.

CRV15. Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. *J. Cryptographic Engineering*, 5(2):73–83, 2015.

DES16. Jack Doerner, David Evans, and Abhi Shelat. Secure stable matching at scale. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1602–1613. ACM Press, October 2016.

DKL+12. Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 241–263. Springer, Heidelberg, September 2012.

DKL+13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, September 2013.

DKS+17. Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *24. Annual Network and Distributed System Security Symposium (NDSS'17)*. The Internet Society, February 26-March 1, 2017. To appear.

DLT14. Ivan Damgård, Rasmus Lauritsen, and Tomas Toft. An empirical study and some improvements of the MiniMac protocol for secure computation. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 398–415. Springer, Heidelberg, September 2014.

DNNR16. Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. Gate-scrambling revisited - or: The TinyTable protocol for 2-party secure computation. Cryptology ePrint Archive, Report 2016/695, 2016. http://eprint.iacr.org/2016/695.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

DR00. Joan Daemen and Vincent Rijmen. Rijndael for AES. In *AES Candidate Conference*, pages 343–348, 2000.

DZ13. Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of Boolean circuits using preprocessing. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 621–641. Springer, Heidelberg, March 2013.

DZ16. Ivan Damgård and Rasmus Winther Zakarias. Fast oblivious AES A dedicated application of the minimac protocol. In *Progress in Cryptology - AFRICACRYPT 2016 - 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings*, pages 245–264, 2016.

EMV17. EMVCo. EMVCo Security QA, 2017. Last accessed February, 2017, https://www.emvco.com/faq.aspx?id=38.

FKOS15. Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 711–735. Springer, Heidelberg, November / December 2015.

GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, Heidelberg, August 2012.

GRR+16. Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 430–443. ACM Press, October 2016.

IKM+13. Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *Proceedings of the 10th Theory of Cryptography Conference on Theory of Cryptography*, TCC'13, pages 600–620, Berlin, Heidelberg, 2013. Springer-Verlag.

JKS08. Somesh Jha, Louis Kruger, and Vitaly Shmatikov. Towards practical privacy for genomic computation. In *2008 IEEE Symposium on Security and Privacy*, pages 216–230. IEEE Computer Society Press, May 2008.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 830–842. ACM Press, October 2016.

KSS13. Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure MPC with dishonest majority. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 549–560. ACM Press, November 2013.

LDDA12. John Launchbury, Iavor S. Diatchki, Thomas DuBuisson, and Andy Adams-Moran. Efficient lookup-table protocol in secure multiparty computation. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 189–200, 2012.

Lit17. Dwayne C Litzenberger. Pycrypto-the python cryptography toolkit. *URL: https://www.dlitz.net/software/pycrypto*, 2017.

LR15. Yehuda Lindell and Ben Riva. Blazing fast 2PC in the offline/online setting with security for malicious adversaries. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 579–590. ACM Press, October 2015.

NISa. Nist: Triple des validation list. http://csrc.nist.gov/groups/STM/cavp/documents/des/tripledesnewval.html. Online; accessed 10-February-2017.

NISb. Nist: Triple des validation list. http://csrc.nist.gov/groups/STM/cavp/documents/aes/aesval.html. Online; accessed 10-February-2017.

NNOB12. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.

NST17. Jesper Buus Nielsen, Thomas Schneider, and Roberto Trifiletti. Constant round maliciously secure 2pc with function-independent preprocessing using lego. In *24th NDSS Symposium*. The Internet Society, 2017. http://eprint.iacr.org/2016/1069.

PSSW09. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009.

PV16. Jürgen Pulkus and Srinivas Vivek. Reducing the number of non-linear multiplications in masking schemes. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 479–497. Springer, 2016.

RP10. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010. Proc.*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.

RR16. Peter Rindal and Mike Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 297–314, 2016.

RV13. Arnab Roy and Srinivas Vivek. Analysis and improvement of the generic higher-order masking scheme of FSE 2012. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013. Proc.*, volume 8086 of *LNCS*, pages 417–434. Springer, 2013.

WMK17. Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. Faster two-party computation secure against malicious adversaries in the single-execution setting. In *EUROCRYPT 2017 Proc.*, 2017.

WRK17. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and communication-efficient, constant-round, secure two-party computation. *IACR Cryptology ePrint Archive*, 2017:30, 2017.

Auxiliary Supporting Material

A Data Encryption Standard and Triple DES

The Data Encryption Standard (DES), was published in 1977 by the the National Institute of Standards and Technology (NIST). It is a block cipher with a key length of 56 bits and block length of 64 bits, based on the design principle of Feistel networks.

A Feistel network proceeds in rounds, as follows. The input of the i-th round is divided into two, 32-bit halves L_{i-1} (left) and R_{i-1} (right). The output is defined as $L_i := R_{i-1}$ and $R_i := L_{i-1} \oplus f_i(k_i, R_{i-1})$, where f_i is a public *keyed round function*. One of the main design advantages of Feistel networks is that the round functions f_i do not need to be invertible. Here, we describe how the 16 round functions $f_i(k_i, R_{i-1})$ of DES are computed by sequentially applying the following steps:

1. `Expansion`: A 48-bit value R'_{i-1} is computed by duplicating some of the bits of R_{i-1}. This has no cost in an arithmetic circuit if each of the bits of R_{i-1} are represented in a different wire.
2. `Key Mixing`: The expanded value is XORed with the 48-bit round key k_i, derived from the global 56-bit key according to some *key schedule*. The output is interpreted as 8 blocks, each of them being 6 bits long.
3. `Substitution`: Each of the 8 blocks is processed with a different S-box, that takes as input 6 bits and has a 4-bit output. These S-Boxes are a non-linear transformation, that can be interpreted either as a polynomial evaluation or as a lookup table.
4. `Permutation`: The output $f_i(R_{i-1})$ is defined as a permutation of the 32 bits obtained in the `Substitution` step.

The result of applying the 16-round Feistel network of DES provides an *encryption* of the 64-bit input. In order to *decrypt*, we can recursively compute the (L_{i-1}, R_{i-1}) halves from (L_i, R_i) by setting $R_{i-1} = L_i$ and then computing $L_{i-1} = L_i \oplus f_i(R_{i-1})$.

The main security drawback of DES is its short key length of 56 bits. A series of practical attacks purely based on exhaustive search, plus the discovery of linear and differential cryptanalysis deemed the standard insecure and led to the standardization of its successors: the Advanced Encryption Standard (AES) in 1997 and the Triple DES in 1999.

Triple DES (officially known as the Triple Data Encryption Algorithm, and usually nicknamed as 3DES) consists of applying DES three successive times using two different keys. More concretely, given 3 56-bit keys k, k_1, k_3, it is defined as $\texttt{3DES}_{k_1,k_2,k_3}(x) := \texttt{DES}_{k_1}(\texttt{DES}^{-1}_{k_2}(\texttt{DES}_{k_3}(x)))$, where $\texttt{DES}^{-1}$ is the *decryption* algorithm of `DES` described above.

Even though 3DES is relatively slow and has only half the block length of AES, it is still a widely deployed standard these days, which motives our study of its efficient evaluation in MPC. At the moment of writing this document,

there are 2362 NIST-validated implementations of Triple DES worldwide [NISa], compared with 4382 for AES [NISb].

B Advanced Encryption Standard

The Advanced Encryption Standard (AES), also known by its original name Rijndael, was declared as a successor of DES by the National Institute of Standards and Technology (NIST) in October 2000 [DR00] after a four-year international competition.

Unlike its predecessor DES, AES does not use a Feistel network. Instead, it relies on a design principle known as a substitution-permutation network (SPN). Each round of an SPN is computed by first XORing the input x with a key k, then applying an invertible *substitution* function S, and finally permuting the bits of the result. The Advanced Encryption Standard is in reality a family of SPN-based block ciphers with different key lengths and number of rounds, but here we will focus on the most widely deployed variant today: the one with a 128-bit key length and 10 iterations over the substitution-permutation network.

Each of these iterations interpret the 128-bit input as a 4-by-4 matrix of bytes called the *state*. During each of them, the `AddRoundKey`, `SubBytes`, `ShiftRows` and `MixColumns` operations are sequentially applied to the state. The only exception is the tenth and last round, which substitutes `MixColumns` with `AddRoundKey`. We describe them in the following:

- `AddRoundKey`: The key is interpreted as a 4-by-4 matrix of bytes and is XORed with the state matrix.
- `SubBytes`: Each byte of the state is replaced with another one according to a lookup table, known as the S-box.
- `ShiftRows`: In this step, the last three rows of the state matrix are cyclically shifted by one, two and three places to the left, respectively. This operation has no cost when represented in an arithmetic circuit, as it only modifies the assignment of the wires for the next step.
- `MixColumns`: This step combines the four bytes in each column of the state, in order to provide diffusion. It can be performed by multiplying a coordinate vector consisting of the four bytes with a circulant MDS matrix in a certain finite field.

The result of applying the substitution-permutation network of AES to an input x is its *encryption*, and *decryption* is computed by inverting the evaluation of the SPN. Note that, given a ciphertext and the key, this is very easy, as the S-box is a public bijection and the MDS matrix used in `MixColumns` is also public.