MPC-Friendly Symmetric Key Primitives

Lorenzo Grassi
Graz University of Technology.

Christian Rechberger
Graz University of Technology.

Dragos Rotaru
Dept. of Comp. Sci.,
University of Bristol.

Peter Scholl
Dept. of Comp. Sci.,
University of Bristol.

Nigel P. Smart
Dept. of Comp. Sci.,
University of Bristol.

ABSTRACT

We discuss the design of symmetric primitives, in particular Pseudo-Random Functions (PRFs) which are suitable for use in a secret-sharing based MPC system. We consider three different PRFs: the Naor-Reingold PRF, a PRF based on the Legendre symbol, and a specialized block cipher design called MiMC. We present protocols for implementing these PRFs within a secret-sharing based MPC system, and discuss possible applications. We then compare the performance of our protocols. Depending on the application, different PRFs may offer different optimizations and advantages over the classic AES benchmark. Thus, we cannot conclude that there is one optimal PRF to be used in all situations.

1. INTRODUCTION

Secure multi-party computation (MPC) allows a set of parties to jointly evaluate a function on private inputs, with the guarantee that no party can learn anything more than the output of the function. In the last decade, MPC has moved from a theoretical pursuit to a very practical field, as protocols have become more efficient and many implementations been been developed.

For many years now, the *de facto* benchmark for MPC systems has been secure computation of the AES function [41, 23, 24, 39, 35]. Although the actual choice of this function was originally as a testbed for comparing protocols, it has often been justified as "useful"; for example if an application needs to evaluate a symmetric encryption scheme or pseudorandom function (PRF) with a secret-shared key. If this is indeed required, then there is no particular reason why AES should be the best choice to work with MPC, compared with other PRFs or symmetric ciphers. Indeed we contend that AES is in many ways a very unnatural choice of a PRF evaluation for use in MPC applications. In this work, we conduct a study of some PRFs for use in MPC, including new protocols for evaluating number-theoretic PRFs, and implementation of "traditional" block cipher candidates designed to have a low complexity in MPC.

CCS'16, October 24 - 28, 2016, Vienna, Austria
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4139-4/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2976749.2978332

1.1 Main Motivating Applications

Before proceeding, we first outline some applications we have in mind. Our focus is on secret sharing based MPC systems such as that typified by BDOZ [7], SPDZ [26, 25], and VIFF [22]; or indeed any classical protocol based on Shamir Secret Sharing. In such situations data is often shared as elements of a finite field $\mathbb{F}_p$, of large prime characteristic. Using such a representation one then has efficient protocols to compute relatively complex functions such as integer comparison [21], fixed point arithmetic [17], and linear programming [16]. Indeed the most famous of such efficient high level protocols is that needed to compute the output of an auction [8].

Given such applications, evaluated by an MPC "engine", the question arises as to how to get data securely in and out of the engine. In traditional presentations the data is entered by the computing parties, and the output is delivered to the computing parties. However, this in practice will be a simplification. Input and output may need to be securely delivered/received by third parties, in addition in a long term reactive functionality the intermediate secure data may need to be stored in a database, or other storage device.

If we examine the case of long term storage of data, which is stored by the MPC engine only to be used again at a later date, the trivial way to store such shared data is for each party to encrypt their share with a symmetric key, and then store each encrypted share. However, this incurs an N-fold increase in storage at the database end (for N MPC servers), which may be prohibitive. A similar trivial solution also applies for data input and output, except data input is now performed using N public keys (one for each MPC server) and output is performed by each server producing a public key encryption of its share to the recipient's public key.

A more efficient solution would be to use a direct evaluation of a symmetric key primitive within the MPC engine. Such a symmetric key primitive should be able to be efficiently evaluated by the MPC engine[1]. We call such a symmetric key primitive "MPC-Friendly". Given almost all symmetric key primitives can be constructed easily from

[1]Note that public key encryption applications as mentioned above can be built from the symmetric key key primitives in the standard KEM-DEM manner. The KEM component being relatively easy to implement, in most cases, in an MPC friendly manner. Thus we focus on symmetric key primitives in this paper.

Pseudo-Random Functions (PRFs), the goal is therefore to produce an MPC-Friendly PRF.

The main problem of using "traditional" PRFs such as AES is that these are built for computational engines which work over data types that do not easily match the operations possible in the MPC engine. For example AES is very much a byte/word oriented cipher, which is hard to represent using arithmetic in $\mathbb{F}_p$. Thus we are led to a whole new area of PRF design, with very different efficiency metrics compared to traditional PRF design.

1.2 Secondary Applications

A simple example application of MPC is to enable distributed secure storage of long-term cryptographic keys, by secret-sharing the key and storing each share at a separate server. When the key is required by an application such as encryption or authentication, the MPC protocol is used to compute this functionality. If this cryptographic functionality is a symmetric cipher, then this application would be greatly enhanced by using an "MPC-Friendly" symmetric primitive.

Using traditional symmetric cryptographic primitives directly on shared data can also improve efficiency for some applications. For example, Laur et al. [33] used an oblivious AES implementation to perform a secure join operation on a secret-shared database. After obliviously shuffling the database, the (deterministic) AES encryptions are made public to all parties, so that the join can then be performed efficiently using standard database algorithms.

Lu and Ostrovsky [36] presented a distributed oblivious RAM protocol, which achieves only $O(\log N)$ overhead, better than any ORAM scheme in the non-distributed setting. This protocol could be combined with a secret-shared MPC system to provide a mechanism to allow secure computation of RAM programs. However, the ORAM construction of [36] makes heavy use of a PRF, so such an application would require the use of an MPC-Friendly PRF.

For other operational reasons it may be useful to encrypt data using a special form of encryption such as deterministic encryption, searchable symmetric encryption (SSE) or (leaky) order-revealing encryption (ORE) [9, 10, 18, 6, 14], under a secret-shared key. These algorithms can enable efficient queries on the encrypted data, whilst the query results can then be decrypted into shares for more complex processing using MPC. For transmission across the wire, to (or from) an external application, a form of Authenticated Encryption (AE) is needed. We note that all of these symmetric primitives (SSE, OPE, AE etc) can be built, in generic ways, out of a PRF. Thus the main obstacle preventing such applications is an efficient MPC-Friendly PRF.

1.3 Related Work

Surprisingly there has been little direct work on this problem, despite the recent plethora of proposed MPC applications; indeed the only paper we know of which explicitly designs PRFs for use in MPC, is [4], which we shall discuss below. The three lines of work most related to the work in this paper, apart from re-purposing designs from elsewhere, are

- Low complexity, "lightweight" ciphers for use in IoT and other constrained environments.
- Block and stream ciphers suited to evaluation by a Fully Homomorphic/Somewhat Homomorphic encryption scheme. So called SHE-Friendly ciphers.
- Designs for use in SNARKs.

We now elaborate on the prior work in these areas.

Low Complexity Lightweight Ciphers: Block ciphers often iterate a relatively simple round function a number of times to achive security goals. Most early designs in this domain focused on small area when implemented as a circuit in hardware. There, large depth (via a large number of rounds) is of no concern as simply means clocking a circuit that implements a single round more times. Notable exceptions are mCrypton[34] and Noekeon[19] which also feature a relatively low depth. The more recent trend to emphasize low latency (with designs like PRINCE[11]) fits much better with our requirement of having low-depth. A property of all these designs is that they lend themselves well to implementations where binary NAND gates, XOR gates, or multiplexers are the basic building blocks in the used libraries. As explained above the majority of secret sharing based MPC applications require description via $\mathbb{F}_p$. Whilst bit operations are possible over $\mathbb{F}_p$ using standard tricks (which alas turn XOR into a non-linear operation), applying such ciphers would require the $\mathbb{F}_p$ data types to be split into a shared bit representation over $\mathbb{F}_p$ to apply the cipher. Such a conversion is expensive.

SHE-Friendly Ciphers: Perhaps due to the recent theoretical interest in SHE/FHE schemes this area has had more attention than the more practical issues addressed in this paper. The motivating scenario for a SHE-Friendly cipher is to enable data to be securely passed to a cloud environment, using a standard encryption scheme, which the cloud server then homomorphically decrypts to obtain a homomorphic encryption of the original data.

This line of work has resulted in a handful of designs. A block cipher called LowMC [4], a stream cipher called Kreyvium [13] (based on the Trivium stream cipher) and FLIP [37] (based on a filter permutation)[2]. The block cipher LowMC is designed for both MPC and FHE implementation, but actually does not meet the MPC design goals we have set. It does indeed have low depth, but it is a cipher based on operations in characteristic two. The two SHE friendly stream cipher designs of Kreyvium and FLIP also suffer from the same problem as the lightweight designs describe above, as they are also bit-oriented.

SNARK-Friendly Constructions: Being SNARK-friendly means that the number of constraints is low. This generally favours larger data types like $\mathbb{F}_p$ or $\mathbb{F}_{2^n}$, and the depth of the circuit is of no concern. MiMC [2] was originally designed for this use case and seems to be the only one in this area. As the depth is not too high either, we choose it for detailed evaluation.

1.4 Contributions

The goal of this work is to investigate the efficient evaluation of PRFs in a secret-sharing based MPC setting. [3] We

[2]FLIP was recently cryptanalysed in [28].

[3]We leave the construction of the various higher level primitives (SSE, ORE, AE etc.) to future work, although many of these can easily be constructed directly from a PRF.

present new protocols for secure computation of PRFs, and implementation results using an actively secure MPC protocol, which tolerates up to $N-1$ out of N corrupted parties (with an online phase based on the SPDZ protocol [26, 25]).

To fix notation we will consider a PRF of the following form

$$F : \begin{cases} (\mathbb{F}_p)^\ell \times (\mathbb{F}_q)^n & \longrightarrow \quad (\mathbb{F}_r)^m \\ (k_1, \dots, k_\ell, x_1, \dots, x_n) & \longmapsto \quad F_k(x_1, \dots, x_n). \end{cases}$$

The various finite fields $\mathbb{F}_p$, $\mathbb{F}_q$ and $\mathbb{F}_r$ may be distinct. Our MPC engine is assumed to work over the finite field $\mathbb{F}_p$, as we always assume the key to the PRF will be a secret shared value. As a benchmark, we compare all of our candidates to the baseline AES example used in prior work, and to implementations of the given PRFs on clear data.

Depending on the precise application, there are several distinct design criteria which we may want to consider. Thus, there will not be a one size fits all PRF which works in all applications. We then have various potential cases:

- In some applications the input is public and we need to embed the public elements $x_1, \dots, x_n \in \mathbb{F}_q$ into $\mathbb{F}_p$. However, the more general case is when the input is secret shared itself, and we have $\mathbb{F}_q = \mathbb{F}_p$.
- In some applications the output of the PRF will be public, and thus $\mathbb{F}_r$ can be any field. In other applications we also want the output to be secret shared, so we can use it in some other processing such as a mode of operation. In this latter case we will have $\mathbb{F}_r = F_p$. In addition, some applications, such as when using the (leaky) ORE scheme presented in [18] require PRF outputs in $\{0, 1, 2\}$, and we may (or may not) require these to be secret shared (and hence embedded in $\mathbb{F}_p$).
- In some applications we would like a PRF which is just efficient in the MPC engine, and we do not care whether the equivalent standard PRF is efficient or not. In other applications we also require that the standard PRF is also efficient. For example when an external third party is encrypting data for the MPC engine to decrypt.

In this paper we consider four candidate PRFs for use in MPC systems, as well as the comparison case of AES. Two of these are number theoretic in nature (the Naor-Reingold PRF, based on DDH, and a PRF based on the Legendre symbol), whilst MiMC [2] and LowMC [4] are more akin to traditional symmetric block cipher constructions.

AES: Since AES does not lend itself well to secure computation over prime fields, we use this purely as a benchmark. We assume an MPC system which is defined over the finite field $\mathbb{F}_{2^8}$, allowing for efficient evaluation of the S-box [23, 24]. We have

$$F_{\mathsf{AES}} : (\mathbb{F}_{2^8})^{16} \times (\mathbb{F}_{2^8})^{16} \to (\mathbb{F}_{2^8})^{16}.$$

LowMC: This is a block cipher candidate [4] designed to be suitable for FHE and MPC style applications; thus it has a low multiplicative depth and a low number of multiplications. It operates over $\mathbb{F}_2$, so like AES, is not well-suited to the MPC applications for which we envisage our block ciphers being used for. Thus we only consider LowMC as an additional base line comparison (along with AES) for our ciphers. LowMC has block size n bits and key size k bits (we use $n = 256$ and $k = 128$), giving:

$$F_{\mathsf{LowMC}} : (\mathbb{F}_2)^k \times (\mathbb{F}_2)^n \to (\mathbb{F}_2)^n$$

Naor-Reingold: Let $\mathbb{G} = \langle g \rangle$ be an elliptic curve group of prime order p in which DDH is hard, and $\mathsf{encode}(\cdot)$ be a hash function that maps elements of $\mathbb{G}$ into elements of $\mathbb{F}_p$. The Naor-Reingold PRF takes a uniform secret-shared key in $\mathbb{F}_p^{n+1}$, a message in $\mathbb{F}_2^n$ (secret-shared over $\mathbb{F}_p$), and outputs a *public* $\mathbb{F}_p$ element as follows:

$$F_{\mathsf{NR}(n)} : (\mathbb{F}_p)^{n+1} \times (\mathbb{F}_2)^n \to \mathbb{F}_p$$
$$(\mathbf{k}, \mathbf{x}) \mapsto \mathsf{encode}(g^{k_0 \cdot \prod_{i=1}^n k_i^{x_i}})$$

To evaluate F_{NR} in MPC naively would require computing exponentiations (or EC scalar multiplications) on secret exponents, which is very expensive. However, if the PRF output is public, we show how the exponentiation (and hence PRF evaluation) can be done very efficiently, with active security, using any MPC protocol based on secret sharing.

Legendre Symbol: We also consider an unusual PRF based on the pseudorandomness of the Legendre symbol. This is a relatively old idea, going back to a paper of Damgård in 1988 [20], but has not been studied much by the cryptographic community. The basic version of the PRF is defined as,

$$F_{\mathsf{Leg(bit)}} : \mathbb{F}_p \times \mathbb{F}_p \to \mathbb{F}_2$$
$$(k, x) \mapsto L_p(x+k)$$

where $L_p(a)$ computes the usual Legendre symbol $\left(\frac{a}{p}\right) \in \{-1, 0, 1\}$ and maps this into $\{0, 1, (p+1)/2\}$, by computing

$$L_p(a) = \frac{1}{2}\left(\left(\frac{a}{p}\right) + 1\right) \pmod p.$$

The output is embedded into $\mathbb{F}_p$, giving a secret-shared output in $\mathbb{F}_p$. If needed, the range can easily be extended to the whole of $\mathbb{F}_p$ by using a key with multiple field elements and performing several evaluations in parallel. This gives a PRF

$$F_{\mathsf{Leg}(n)} : (\mathbb{F}_p)^{((n+1)\cdot\ell)} \times (\mathbb{F}_p)^n \to \mathbb{F}_p,$$

for some value $\ell = O(\log_2 p)$ chosen large enough to ensure a sufficient statistical distance from uniform of the output. This PRF takes n finite field elements as input and produces an element in $\mathbb{F}_p$ as output, where n is some fixed (and relatively small) number, say one or two.

Perhaps surprisingly, we show that the Legendre PRF can be evaluated *very efficiently* in MPC, at the cost of just two multiplications in three rounds of interaction for $F_{\mathsf{Leg(bit)}}$. To the best of our knowledge, this is the only PRF that can be evaluated in a constant number of rounds on secret-shared data, using any arithmetic MPC protocol. Since the underlying hard problem is less well-studied than, say, DDH or factoring, we also provide a brief survey of some known attacks, which are essentially no better than brute force of the key.

MiMC: This is a very recent class of designs whose primary application domain are SNARKs [2]. In addition to a cryptographic hash function, the design also includes a block cipher which is also usable as a PRF, with up to birthday bound security. The input, output and keys are all defined over $\mathbb{F}_p$, so we get:

$$F_{\mathsf{MiMC}} : \mathbb{F}_p \times \mathbb{F}_p \rightarrow \mathbb{F}_p.$$

The core of the round function is the simple map $x \mapsto x^3$ over $\mathbb{F}_p$. The number of rounds is quite high (for a 128-bit prime p 82 for full security, 73 for PRF security), but in terms of $\mathbb{F}_p$ multiplications the performance turns out to be competitive.

The reason for selecting MiMC as a "standard" block cipher is that firstly it works over a finite prime field of large characteristic, which is a common requirement for applications of secret-sharing based MPC that perform arithmetic on integers or fixed-point data types. Secondly, the depth of the computation is not too large, being 146. Thirdly, the number of non-linear operations is also 146, this means that the offline pre-processing needed (to produce multiplication triples) will be very small compared to other constructions.

In Table 1 we present an overview of the MPC-friendly PRFs we consider. The table shows the number of secure multiplication needed to execute the online evaluation of the function on shared inputs (since in secret-sharing based MPC, additions are free) as well as the number of rounds of communication.

1.5 Length Extension

We end this introduction by noting that F_{MiMC} and $F_{\mathsf{Leg}(n)}$ can be extended to cope with arbitrary length inputs in the standard way; either by using a CBC-MAC style construction or a Merkle–Damgård style construction. For example, to extend $F_{\mathsf{Leg}(1)}$ and F_{MiMC}, so that they can be applied to an input $x_1, \ldots, x_n \in \mathbb{F}_p$ we can use CBC mode as in Figure 1. Whereas, to extend $F_{\mathsf{Leg}(2)}$ we can apply Merkle–Damgard as in Figure 2. These two extension techniques are often more efficient than using an arbitrary length PRF as a base building block.

Length Extension for $F_{\mathsf{Leg}(1)}$

```
1. $c_0 \leftarrow n$.
2. For $i = 1, \ldots, n$ do
    (a) $c_i \leftarrow x_i + F_{\mathsf{Leg}(1)}(k, c_{i-1})$.
3. $a \leftarrow F_{\mathsf{Leg}(1)}(k, c_n)$.
4. Return $a$.
```

Figure 1: Using CBC Mode With $F_{\mathsf{Leg}(1)}$

Length Extension for $F_{\mathsf{Leg}(2)}$

```
1. $c_0 \leftarrow n$.
2. For $i = 1, \ldots, n$ do
    (a) $c_i \leftarrow F_{\mathsf{Leg}(2)}(k, c_{i-1}, x_i)$.
3. Return $c_n$.
```

Figure 2: Using Merkle-Damgård With $F_{\mathsf{Leg}(2)}$

2. BACKGROUND

In this section we outline some of the basic material which we will assume for the rest of this paper.

2.1 Multi-Party Computation Model

The general model of MPC we consider is the so-called *arithmetic black box*, which is an ideal functionality that allows parties to input and output values to be secret-shared, and performs basic arithmetic operations on these secret values over a finite field $\mathbb{F}_p$. This abstracts away the underlying details of secret-sharing and MPC, and gives us the commands in Figure 3. Note that as well as addition and multiplication, $\mathcal{F}_{\mathsf{ABB}}$ has commands for generating random values according to various distributions, which allows more efficient protocols for certain tasks. Finally, the **Share** command gives parties access to random, additive shares of a value stored in the box. This essentially assumes the underlying MPC protocol uses additive secret sharing, but is only used for the Naor-Reingold PRF protocol (Section 3).

We use the notation $[x]$ to denote a secret-shared value that is stored in $\mathcal{F}_{\mathsf{ABB}}$. We also define addition and multiplication operators for the $[\cdot]$ notation; so, for example, the statement

$$[w] = [x] \cdot [y] + 2[z]$$

implicitly means that the **Add** and **Mult** commands of $\mathcal{F}_{\mathsf{ABB}}$ are used to compute the shared value $[w]$.

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

Each value stored in this functionality is associated with a unique identifier that is given to all parties. Let $[x]$ denote the identifier for a value x that is stored by the functionality. Let $A \subset \{1, \ldots, n\}$ denote the index set of corrupted parties.

Input: Receive a value $x \in \mathbb{F}_p$ from some party and store x.

Add$([x], [y])$**:** Compute $z = x + y$ and store $[z]$.

Mult$([x], [y])$**:** Compute $z = x \cdot y$ and store $[z]$.

Random: Sample $r \overset{R}{\leftarrow} \mathbb{F}_p$ and store $[r]$.

RandomBit: Sample $b \overset{R}{\leftarrow} \{0, 1\} \subset \mathbb{F}_p$ and store $[b]$.

RandomSquare: Sample $s \overset{R}{\leftarrow} \mathbb{F}_p$ and store $[s^2]$.

Share$([x])$**:** For each $i \in A$, receive $x_i \in \mathbb{F}_p$ from the adversary. Sample uniform honest parties' shares $\{x_j\}_{j \notin A}$, subject to the constraint that $\sum_{i=1}^{n} x_i = x$. Send x_i to party P_i.

Output$([x])$**:** Send the value x to all parties.

Figure 3: Ideal functionality for arithmetic MPC

Concretely, the MPC protocol we use to implement $\mathcal{F}_{\mathsf{ABB}}$ is the SPDZ protocol by Damgård et al. [26, 25], which operates over a finite field of size $\geq 2^\kappa$ (for statistical security κ) and provides active security against any number of corrupted parties. The protocol consists of two stages: a *preprocessing* phase, which is independent of the inputs and done in advance, and a more efficient *online* phase, where the actual computation takes place. The purpose of the preprocessing is to generate enough random shared data that will be consumed later by the online phase. The main data produced is one of three different forms:

- **Multiplication Triples:** A triple of random shares $([a], [b], [c])$ such that $c = a \cdot b \mod p$.

PRF	$\log_2 p$	Output (type)	Online cost		Assumption
			Mult.	Rounds	
F_{AES}	8	shared	960	50	–
F_{LowMC}	2	shared	1911	13	–
$F_{\mathsf{NR}}(n)$	256	public	$2 \cdot n$	$3 + \log(n+1)$	EC-DDH
$F_{\mathsf{Leg}}(\mathsf{bit})$	128	shared	2	3	DSLS
$F_{\mathsf{Leg}}(n)$	128	shared	$256 \cdot n$	3	DSLS
F_{MiMC}	128	shared	146	73	–

Table 1: Overview of the cost of evaluating the PRFs in MPC.

- **Square Pairs:** A pair of random shares $([a], [b])$ such that $b = a^2 \mod p$.
- **Random Bit:** A random sharing $[a]$ of a value $a \in \{0, 1\}$.

The main arithmetic operations in $\mathcal{F}_{\mathsf{ABB}}$ have roughly the following complexity when implemented in SPDZ. Additions (and linear operations) are local operations so essentially for free. A multiplication uses a preprocessed multiplication triple and requires sending two field elements in the online phase, with one round of interaction. Squaring can be done using a square pair and sending just one field element, again in one round.

The preprocessing can be implemented using either somewhat homomorphic encryption (SHE) — as in the original SPDZ protocols — or oblivious transfer (OT), using the recent protocol of Keller et al. [31]. We present runtimes using the OT-based offline phase only, as it is much more efficient, even when compared with the weaker covertly secure protocols using SHE.

2.2 MPC Evaluation of AES and LowMC

As a means of comparison for the other PRFs we use as a base line a two party implementation of AES using a SPDZ engine over the finite field $\mathbb{F}_{2^8}$, embedded into $\mathbb{F}_{2^{40}}$, as in [24]. We estimate the offline phase costs 200ms per block, with an online phase latency of 8ms and a throughput of over 500 blocks per second. Note that recently, much lower latencies have been obtained by evaluating AES using secure table lookup [27]. However, this technique requires far more (i.e., 256 times) preprocessing data, so we do not consider this. One should also bear in mind that this is only the time needed to evaluate the PRF. In a given application, which is likely to be over a different finite field, the MPC engine will also need to convert data between the two fields $\mathbb{F}_p$ and $\mathbb{F}_{2^{40}}$. This is likely to incur a more significant cost than the evaluation of the PRF itself.

In addition to AES, we also present comparison executions for the low complexity block cipher LowMC. This is to enable a comparison with our $\mathbb{F}_p$ based block ciphers against not only a standard in-use block cipher (AES), but also a block cipher designed for use in MPC/FHE environments.

2.2.1 F_{LowMC} *Definition*

LowMC [4] is a flexible family of block ciphers with operations over $\mathbb{F}_2$, designed to have a low number of multiplications and a low multiplicative depth when implemented in MPC. Similar to AES, it is based on an SPN structure where the block size n, the key size k, the number of Sboxes m in the substitution layer and the allowed data complexity d of attacks can independently be chosen. The number of rounds r needed to reach the security claims is then derived from these parameters. The two most relevant parts of the round transformation are the SboxLayer and the LinearLayer. SboxLayer is an m-fold parallel application of the same 3-bit Sbox (of multiplicative depth 1) on the first $3m$ bits of the state. If $n > 3m$ then for the remaining $n - 3m$ bits, the SboxLayer is the identity. LinearLayer is the multiplication in $\mathbb{F}_2$ of the state with a predetermined dense randomly chosen invertible binary $n \times n$ matrix that is different for every round.

Using the most recent v2[3] formula for r, we need at least 13 rounds to achieve a security comparable to AES as a PRF, i.e. $k = 128$ and $d = 64$. Using $n = 256$, the minimal number Sboxes m for which this is true turns out to be 49.

2.2.2 *Computing* F_{LowMC} *in MPC*

To evaluate LowMC in MPC, we consider two approaches. In the first method, denoted $F_{\mathsf{LowMC}}(\mathsf{vector})$, we work over $\mathbb{F}_{2^{128}}$ and compute the matrix multiplications and XOR operations by parallelizing over 128-bit vectors. Specifically, each column M_i of the $n \times n$ matrix M is packed into $\mathbb{F}_{2^{128}}$ elements; to compute the product $M[\mathbf{x}]$ we take the inner product of all columns with $[\mathbf{x}]$. For $n = 256$, this requires 512 XORs and 512 local finite field multiplications. However, we then need to switch back to $\mathbb{F}_2$ to evaluate the Sbox (with three $\mathbb{F}_2$ multiplications), which requires bit decomposition, adding one round of interaction for every round of the cipher.

In the second approach, denoted $F_{\mathsf{LowMC}}(\mathsf{M4R})$, we use the "Method of Four Russians" [1] to perform each matrix multiplication in $O(n^2/\log n)$ bit operations. We do not parallelize the computation by packing bits into vectors, so this actually results in a higher computation cost than the vector method, but avoids the need for bit decomposition in each round.

In both methods, the total number of multiplications over $\mathbb{F}_2$ is $3 \cdot m \cdot r$. The vector approach requires $256 \cdot r$ additional random bits, and also $2r$ rounds of communication, instead of r rounds for M4R.

2.2.3 *Performance*

With parameters $n = 256, m = 49, r = 13$, we obtained a latency of 4ms and a throughput of almost 600 blocks per second.

As for AES, the need to convert from a $\mathbb{F}_p$ representation to a bit-oriented representation for application of LowMC is likely to dominate the run-time for the actual PRF evaluation, making LowMC unsuitable for the applications we discussed at the beginning.

3. NAOR–REINGOLD PRF

In this section we describe the Naor-Reingold PRF, originally presented in [38]. We then go on to describe how it can be efficiently implemented in a secret sharing based MPC system.

3.1 F_{NR} Definition

Let $\mathbb{G} = \langle g \rangle$ be a multiplicatively written group of prime order p in which DDH is hard, and $\mathsf{encode}(\cdot)$ be a hash function that maps group elements into elements of $\mathbb{F}_p$. For a message $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$, the Naor-Reingold PRF [38] is defined by:

$$F_{\mathsf{NR}(n)}(\mathbf{k}, \mathbf{x}) = \mathsf{encode}(g^{k_0 \cdot \prod_{i=1}^{n} k_i^{x_i}})$$

where $\mathbf{k} = (k_0, \ldots, k_n) \in \mathbb{F}_p^{n+1}$ is the key.

In practice, we choose $\mathbb{G}$ to be a 256-bit elliptic curve group over the NIST curve P-256, so require an MPC protocol for $\mathbb{F}_p$ with a 256-bit prime p.

3.2 Public Output Exponentiation Protocol

The main ingredient of our method to evaluate F_{NR} in MPC, when the key and message are secret-shared over $\mathbb{F}_p$, is an efficient protocol for publicly computing g^s, for some secret value $s \in \mathbb{F}_p$. The protocol, shown in Figure 4, uses any arithmetic MPC protocol based on linear secret sharing over $\mathbb{F}_p$. This is modeled for the case of additive secret sharing by the **Share** command of the $\mathcal{F}_{\mathsf{ABB}}$ functionality, which produces random shares of secret values.

Given additive shares $s_i \in \mathbb{F}_p$, each party P_i first broadcasts g^{s_i}, so the result $y = \prod g^{s_i}$ can be computed. To obtain active security, we must ensure that each party used the correct value of s_i. We do this by computing an additional public exponentiation of g^t, where $t = r \cdot s$ for some random, secret value r. This serves as a one-time MAC on s, which can then be verified by opening r and checking that $g^t = y^r$. If an adversary cheats then passing the check essentially requires guessing the value of r, so occurs only probability $1/p$.

Note that the functionality $\mathcal{F}_{\mathsf{ABB\text{-}Exp}}$ (Figure 5) models an unfair computation, whereby the adversary first learns the output, and can then decide whether to give this to the honest parties or not. This is because in the protocol, they can always simply stop sending messages and abort after learning y.

Theorem 1. *The protocol Π_{Exp} securely computes the functionality $\mathcal{F}_{\mathsf{ABB\text{-}Exp}}$ in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model.*

Proof. We construct a simulator $\mathcal{S}$, which interacts with any adversary $\mathcal{A}$ (who controls the corrupt parties $\{P_i : i \in A\}$) and the ideal functionality $\mathcal{F}_{\mathsf{ABB\text{-}Exp}}$, such that no environment can distinguish between an interaction with $\mathcal{S}$ and a real execution of the protocol Π_{Exp}.

- In the first round $\mathcal{S}$ receives s_i for $i \in A$, as the corrupt parties' inputs to the $\mathcal{F}_{\mathsf{ABB\text{-}Share}}$ command. $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{ABB\text{-}Exp}}$ with $(\mathtt{exp}, [s])$ and receives $y = g^s$. Then $\mathcal{S}$ samples $s_i \xleftarrow{R} \mathbb{F}_p$ and sets $y_i = g^{s_i}$ for all $i \notin A$. $\mathcal{S}$ modifies one honest party's share y_i to $g^s \prod_{j \neq i} y_j^{-1}$, then sends y_i for all $i \notin A$ to the adversary and gets back the corrupted parties' response y_i^*, for $i \in A$.
- Proceed similarly to the previous step: $\mathcal{S}$ samples $r_i \xleftarrow{R} \mathbb{F}_p$, sets $z_i = y_i^{r_i}$ such that $\prod_i z_i = y^r$. Sends z_i to $\mathcal{A}$ on behalf of the honest parties. Receives back courrupted parties z_i^*.
- Sends $r \leftarrow \sum_i r_i$ to the adversary. $\mathcal{S}$ performs the checking phase with z_i^* from $\mathcal{A}$ and the honest z_i. If the check passes send **Deliver** to $\mathcal{F}_{\mathsf{ABB\text{-}Exp}}$.

The indistinguishability argument follows from the fact that all broadcasted values g^{x_i} by $\mathcal{S}$ and the real protocol Π_{Exp} have uniform distribution over $\mathbb{F}_p$ with output in $\mathbb{G}$ with respect to $\prod_i g^{x_i} = g^x$.

Correctness is straightforward if all parties follow the protocol. An adversary $\mathcal{A}$ wins if it changes the distribution of the functionality to output **Deliver**. Alas, this happens with negligible probability:

Suppose a corrupt party P_j sends y_j^* instead of $y_j = g^{s_j}$. We can write $y_j^* = g^{s_j} \cdot e$, for some error $e \neq 1 \in \mathbb{G}$, and so $y = g^s \cdot e$. Then the check passes if $\mathcal{A}$ can come up with z_j^* such that $\prod_i z_i = g^{rs} \cdot e^r$. Writing $z_j^* = z_j \cdot f$, this is equivalent to coming up with $f \in \mathbb{G}$ such that $f = e^r$. Since r is uniformly random and unknown to the adversary at the time of choosing e and f, passing this check can only happen with probability $1/|\mathbb{G}|$. Note that this requires $\mathbb{G}$ to be of prime order, so that e (which is adversarially chosen) is always a generator of $\mathbb{G}$. □

Protocol $\Pi_{\mathsf{Exp}}([s])$

1. The parties call $\mathcal{F}_{\mathsf{ABB}}$ with command (**Share**, $[s]$), so that each party P_i obtains an additive share $s_i \in \mathbb{F}_p$
2. Each party P_i broadcasts $y_i = g^{s_i}$
3. Compute $y = \prod_i y_i$
4. Take a random shared $[r]$, and compute $[t] = [r] \cdot [s]$
5. Call $\mathcal{F}_{\mathsf{ABB}}$ with (**Share**, $[t]$) so that each P_i obtains t_i. Broadcast $z_i = g^{t_i}$
6. Open $[r]$ and check that $\prod_i z_i = y^r$
7. Output y

Figure 4: Securely computing a public exponentiation

Functionality $\mathcal{F}_{\mathsf{ABB\text{-}Exp}}$

Let $G = \langle g \rangle$ be a group of prime order p. This functionality has all of the features of $\mathcal{F}_{\mathsf{ABB}}$ (running in $\mathbb{F}_p$), plus the following command:

PubExp: On receiving $(\mathtt{exp}, [s])$ from all parties, where s is stored in memory, retrieve s, then send $y = g^s$ to the adversary and wait for a response. If the adversary responds with **Deliver** then send y to all parties. Otherwise output $\perp$ to all parties.

Figure 5: Ideal functionality for public exponentiation

More Efficient Protocol based on SPDZ.

When using the SPDZ MPC protocol with the secret-shared MAC representation from [25], we can save performing the multiplication $[t] = [r] \cdot [s]$. Instead, we can take the shared MAC value $[m]$ (on the shared s), which satisfies $m = s \cdot \alpha$ for a shared MAC key α, and use $[m]$ and $[\alpha]$ in place of $[t]$ and $[r]$. However, in this case α cannot be made public, otherwise all future MACs could be forged. Instead, steps 4–6 are replaced with:

- Each party commits to $z_i = y^{\alpha_i} \cdot g^{-m_i}$.
- All parties open their commitments and check that $\prod_i z_i = 1$.

If the parties are honest, we have $z_i = g^{s \cdot \alpha_i - m_i}$, so the check will pass. Since in SPDZ, the honest parties' MAC shares m_i are uniformly random, the shares of α_i are perfectly masked by the g^{-m_i} factor in z_i, so no information on α is leaked. The main difference here is that the parties must commit to the z_i shares before opening, to prevent a rushing adversary from waiting and forcing the product to always be 1. The number of rounds and exponentiations is the same, but one multiplication is saved compared with the previous protocol.

3.3 Secure Computation of Naor-Reingold

Protocol Π_{NR}

KeyGen: Call $\mathcal{F}_{\mathsf{ABB}}$.Random to generate $n+1$ random keys $[k_0] \dots [k_n]$.

Eval: To evaluate $F_{\mathsf{NR}(n)}(k, x)$ on input $[x]$ with key $[k]$:

1. Bit decompose $[x]$ into $[x_1] \dots [x_n]$.
2. Compute $[s] = [k_0] \cdot \prod_{i=1}^{n}([k_i][x_i] + (1 - [x_i])$ (see text for details).
3. Call $\mathcal{F}_{\mathsf{ABB\text{-}Exp}}$ on input $[s]$.

Figure 6: Computing $F_{\mathsf{NR}(n)}(\mathbf{k}, \mathbf{x})$

Given the protocol for public exponentiation, it is straightforward to evaluate the Naor-Reingold PRF with public output when given a bit-decomposed, secret-shared input $[x_1], \dots, [x_n]$ and key $[k_0], \dots, [k_n]$. First compute

$$[s] = [k_0] \cdot \prod_{i=1}^{n}([x_i] \cdot [k_i] + (1 - [x_i]))$$

using $\mathcal{F}_{\mathsf{ABB}}$, and then use Π_{Exp} to obtain g^s.

The product can be computed in $\lceil \log_2 n + 1 \rceil$ rounds using a standard binary tree evaluation. Alternatively, we can obtain a constant (4) rounds protocol using the prefix multiplication protocol of Catrina and de Hoogh [15], (which is an optimized variant of the trick of Bar-Ilan and Beaver [5]) at the expense of $2(n+1)$ additional multiplications.

Security of the Π_{NR} protocol is straightforward, since there is no interaction outside of the arithmetic black box functionality.

Handling Input in $\mathbb{F}_p$.

If the input is given as a field element rather than in bit-decomposed form, then we must first run a bit decomposition protocol, such as that of Catrina and de Hoogh [15] or Damgård et al. [21]. The latter works for arbitrary values of x, whilst the former is more efficient, but requires x is ℓ bits long, where $p > 2^{\ell+\kappa}$ for statistical security κ.

Complexity.

For the logarithmic rounds variant based on SPDZ, with n-bit input that is already bit decomposed, the protocol requires $2n$ multiplications of secret values and three exponentations, in a total of $\lceil \log_2 n + 1 \rceil + 3$ rounds. The constant rounds variant takes $4n+2$ multiplications in 7 rounds. Note that there is a higher cost for the secure multiplications, as we require an MPC protocol operating over $\mathbb{F}_p$ for a 256-bit prime p (for 128-bit security), whereas our other PRF protocols only require MPC operations in 128-bit fields.

3.4 Performance

The main advantage of this PRF is the small number of rounds required, which leads to a low latency in our benchmarks (4.4ms over LAN). However, the high computation cost (for EC operations) slows down performance and results in a low throughput. We found that with a 256-bit prime p and $n = 128$, the logarithmic rounds variant outperformed the constant rounds protocol in all measures in a LAN environment. In a WAN setting, the constant round protocol achieves a lower latency, but is worse for throughput and preprocessing time.

4. PRF FROM THE LEGENDRE SYMBOL

In this section we consider a PRF based on the Legendre symbol, which to the best of our knowledge was first described in [44]. Whilst this PRF is very inefficient when applied to cleartext data, we show that with secret-shared data in the MPC setting it allows for a very simple protocol.

4.1 F_{Leg} Definition

In 1988, Damgård proposed using the sequence of Legendre symbols with respect to a large prime p as a pseudorandom generator [20]. He conjectured that the sequence

$$\left(\frac{k}{p}\right), \left(\frac{k+1}{p}\right), \left(\frac{k+2}{p}\right), \dots$$

is pseudorandom, when starting at a random seed k. Although there have been several works studying the *statistical* uniformity of this sequence, perhaps surprisingly, there has been very little research on cryptographic applications since Damgård's paper. Damgård also considered variants with the Jacobi symbol, or where p is secret, but these seem less suitable for our application to MPC.

We first normalize the Legendre symbol to be in $\{0, 1, (p+1)/2\}$, by defining:

$$L_p(a) = \frac{1}{2}\left(\left(\frac{a}{p}\right) + 1\right) \pmod p.$$

We now define the corresponding pseudorandom function (as in [44]) as

$$F_{\mathsf{Leg(bit)}}(k, x) = L_p(k + x)$$

for $k, x \in \mathbb{F}_p$, where $p \approx 2^\lambda$ is a public prime. The security of this PRF is based on the following two problems:

Definition 1 (Shifted Legendre Symbol Problem)**.** *Let k be uniformly sampled from $\mathbb{F}_p$, and define $\mathcal{O}_{\mathsf{Leg}}$ to be an oracle that takes $x \in \mathbb{F}_p$ and outputs $\left(\frac{k+x}{p}\right)$. Then the* Shifted Legendre Symbol *(SLS) problem is to find k, with non-negligible probability.*

Definition 2 (Decisional Shifted Legendre Symbol Problem). *Let $\mathcal{O}_{\mathsf{Leg}}$ be defined as above, and let $\mathcal{O}_{\mathsf{R}}$ be a random oracle that takes values in $\mathbb{F}_p$ and produces outputs in $\{-1, 1\}$. The* Decisional Shifted Legendre Symbol *(DSLS) problem is to distinguish between $\mathcal{O}_{\mathsf{Leg}}$ and $\mathcal{O}_{\mathsf{R}}$ with non-negligible advantage.*

The following proposition is then immediate.

Proposition 1. *The function $F_{\mathsf{Leg(bit)}}$ is a pseudorandom function if there is no probabilistic polynomial time algorithm for the DSLS problem.*

4.2 Hardness of the Shifted Legendre Symbol Problem

The SLS problem has received some attention from the mathematical community, particularly in the quantum setting. We briefly survey some known results below.

A naive algorithm for *deterministically* solving the SLS problem is to compute $\left(\frac{k+x}{p}\right)$ for all $(k, x) \in \mathbb{F}_p^2$ and compare these with $\mathcal{O}_{\mathsf{Leg}}(x)$ for all $x \in \mathbb{F}_p$, which requires $\tilde{O}(p^2)$ binary operations. Russell and Shparlinski [42] described a more sophisticated algorithm using Weil's bound on exponential sums, which reduces this to $\tilde{O}(p)$.

Van Dam, Hallgren and Ip [44] described a quantum polynomial time algorithm for the SLS problem that recovers the secret k if the oracle can be queried on a quantum state. They conjectured that classically, there is no polynomial time algorithm for this problem. Russell and Shparlinski [42] also extended this quantum algorithm to a generalization of the problem where the secret is a polynomial, rather than just a linear shift.

One can also consider another generalization called the *hidden shifted power problem*, where the oracle returns $(k + x)^e$ for some (public) exponent $e|(p-1)$. The SLS problem is a special case where $e = (p-1)/2$. Vercauteren [45] called this the *hidden root problem* and described efficient attacks over small characteristic extension fields, with applications to fault attacks on pairings-based cryptography. Bourgain et al. [12] showed that if $e = p^{1-\delta}$ for some $\delta > 0$ then this problem has classical query complexity $O(1)$. Note that neither of these attacks apply to the SLS problem, which cannot be solved with fewer than $\Omega(\log p)$ queries [43].

In conclusion, we are not currently aware of any classical algorithms for the SLS problem in better than $\tilde{O}(p)$ time, nor of any method for solving the DSLS problem without first recovering the secret. We note that unlike discrete log and factoring, it is still an open question as to whether there are even efficient quantum algorithms if the SLS oracle can only be queried classically.

4.3 Secure Computation of $F_{\mathsf{Leg(bit)}}$

It turns out that $F_{\mathsf{Leg(bit)}}$ can be evaluated in MPC very efficiently, at roughly the cost of just 2 multiplications in 3 rounds of communication. Although this only produces a single bit of output, composing together multiple instances in parallel with independent keys allows larger outputs to be obtained (see later).

We first describe how to evaluate $F_{\mathsf{Leg(bit)}}$ when the output is public, and then show how to extend this to secret-shared output, with only a small cost increase.

Public output.

Suppose we have a shared, non-zero $[a]$ and want to compute the public output, $L_p(a)$. Since the output is public, we can simply take a random preprocessed non-zero square $[s^2]$, compute $[c] = [s^2] \cdot [a]$ and open c. By the multiplicativity of the Legendre symbol, $L_p(c) = L_p(a)$.

By composing the PRF n times in parallel, this gives an n-bit output PRF that we can evaluate in MPC with just n multiplications and n openings in two rounds. The preprocessing requires n random squares and multiplication triples.

Shared output.

Now suppose we instead want shared output, $[L_p(a)]$. If we have a random non-zero value $[t]$, and also the shared value $[L_p(t)]$, then this is easy. Just open $[a] \cdot [t]$, and compute the Legendre symbol of this to get $c = L_p(a \cdot t)$. The shared value $[L_p(a)]$ can then be computed locally using c and $[L_p(t)]$, as c is public.

Generating a random value with a share of its Legendre symbol can be done very cheaply. Our key observation is that we can do this without having to compute *any* Legendre symbols in MPC. Let $\alpha \in \mathbb{Z}_p$ be a (public) quadratic non-residue, and perform the following:

- Take a random square $[s^2]$ and a random bit $[b]$.
- Output $(2[b] - 1, [b] \cdot [s^2] + (1 - [b]) \cdot \alpha \cdot [s^2])$

Note that since α is a non-square, the second output value is clearly either a square or non-square based on the value of the random bit b (which is mapped into $\{-1, 1\}$ by computing $2 \cdot b - 1$). Finally, note that since s^2 provides fresh randomness each time, α can be reused for every PRF evaluation. This gives us the protocol in Figure 7, which realizes the functionality $\mathcal{F}_{\mathsf{ABB\text{-}Leg}}$ shown in Figure 8. Notice that all bar the computation of u can be performed in a preprocessing phase if needed.

Protocol Π_{Legendre}

Let α be a fixed, quadratic non-residue modulo p.

KeyGen: Call $\mathcal{F}_{\mathsf{ABB}}$.Random to generate a random key $[k]$.

Eval: To evaluate $F_{\mathsf{Leg(bit)}}$ on input $[x]$ with key $[k]$:

1. Take a random square $[s^2]$ and a random bit $[b]$
2. $[t] \leftarrow [s^2] \cdot ([b] + \alpha \cdot (1 - [b]))$
3. $u \leftarrow \mathsf{Open}([t] \cdot ([k] + [x]))$
4. Output $[y] \leftarrow ((\frac{u}{p}) \cdot (2[b] - 1) + 1)/2$

Figure 7: Securely computing the $F_{\mathsf{Leg(bit)}}$ PRF with secret-shared output

Security.

At first glance, the security of the protocol appears straightforward: since t and k are uniformly random, the opened value u should be simulatable by a random value, and this will be correct except with probability $1/p$ (if $s^2 = 0$). However, proving this turns out to be more tricky. We need to take into account that if $x = -k$ then the protocol causes $u = 0$ to be opened, but in the ideal world the simulator does not know k so cannot simulate this. This reflects the fact that an adversary who solves the SLS problem can find k and run the protocol with $x = -k$. Therefore, we need to assume hardness of the SLS problem and show that any environment that distinguishes the two worlds (by causing $x = -k$ to be queried) can be used to recover the key k. The reduction must use the SLS oracle, $\mathcal{O}_{\mathsf{Leg}}$, to detect whether $x = -k$, in order to simulate the u value to the environment. To do this, they simply obtain the value $y = \left(\frac{x+k}{p}\right)$ from $\mathcal{O}_{\mathsf{Leg}}$ and check whether $y = 0$, for each **Eval** query made by the adversary.

Functionality $\mathcal{F}_{\mathsf{ABB\text{-}Leg}}$

This functionality has all of the same commands as $\mathcal{F}_{\mathsf{ABB}}$, plus the following:

KeyGen: On receiving (`keygen`) from all parties, sample $k \xleftarrow{R} \mathbb{F}_p$ and store k.

PRF: On receiving (`legendre`, $[x]$) from all parties, where x is stored in memory, compute $y = L_p(x + k)$ and store y in memory.

Figure 8: Ideal functionality for the Legendre symbol PRF, $F_{\mathsf{Leg(bit)}}$

Theorem 2. *The protocol Π_{Legendre} securely computes the functionality $\mathcal{F}_{\mathsf{ABB\text{-}Leg}}$ in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model, if the SLS problem is hard.*

Proof. We construct a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ corrupting up to $n - 1$ parties can distinguish between the real protocol Π_{Legendre}, and $\mathcal{S}$ interacting with the ideal functionality $\mathcal{F}_{\mathsf{ABB\text{-}Leg}}$.

In the **KeyGen** stage, $\mathcal{S}$ simply calls $\mathcal{F}_{\mathsf{ABB\text{-}Leg}}$ with the `keygen` command. In the **Eval** stage, the main task of $\mathcal{S}$ is to simulate the opened value u, which is done by sampling $u \xleftarrow{R} \mathbb{F}_p$, and then call $\mathcal{F}_{\mathsf{ABB\text{-}Leg}}$ with (`legendre`, $[x]$).

We now argue indistinguishability of the two executions. In the real world, since t is computed as $s^2 \cdot (b + (1 - b) \cdot \alpha)$ for a uniform quadratic residue s^2 and random bit b, then t is uniform in $\mathbb{F}_p$. This is because the map defined by multiplication by α is a bijection between the sets of squares and non-squares modulo p. Therefore, if s^2 is a uniformly random square, then $\alpha \cdot s^2$ is a uniformly random non-square.

Now, since t is a fresh uniformly random value on each evaluation, the real world value u and output y, as seen by $\mathcal{Z}$, will be identically distributed to the simulated values as long as $k + x \neq 0$ and $s \neq 0$. Whenever the former happens in the real world $u = 0$ is opened, whereas the ideal world still simulates a random value, so the environment can distinguish. In the latter case, $s = 0$, the output y will be incorrectly computed in the real world, but this can only happen with probability $1/p$.

However, any environment $\mathcal{Z}$ that causes $k+x = 0$ to happen with non-negligible probability can be used to construct an algorithm $\mathcal{A}^*$ that breaks the SLS problem, as follows.

$\mathcal{A}^*$ runs $\mathcal{Z}$, emulating a valid execution of Π_{Legendre} by replacing $L_p(x + k)$ computation with calls to $\mathcal{O}_{\mathsf{Leg}_k}$. These modified transcripts have the same distribution since the SLS oracle and (`keygen`) both generate a random key. When $\mathcal{A}^*$ runs $\mathcal{Z}$ internally, it knows the inputs provided by $\mathcal{Z}$ to all parties, so knows the x value on each invocation of Π_{Legendre}. Once $\mathcal{Z}$ constructs a query for which $\mathcal{O}_{\mathsf{Leg}_k}$ returns 0 then $\mathcal{A}^*$ responds to the SLS challenge with $k = -x$. Finally, the algorithm looks like this:

1. Interact with $\mathcal{Z}$ as the simulator $\mathcal{S}$ would do.
2. Instead of computing the Legendre symbol $L_p(x + k)$ as in $\mathcal{F}_{\mathsf{ABB\text{-}Leg}}$, make a call to $\mathcal{O}_{\mathsf{Leg}_k}$.
3. If $\mathcal{O}_{\mathsf{Leg}_k}(x) = 0$, return $-x$ as the SLS secret.

The only way $\mathcal{Z}$ can distinguish between $\mathcal{S}$ and Π_{Legendre} — except with probability $1/p$ — is by producing a query x for which $\mathcal{O}_{\mathsf{Leg}_k}(x) = 0$, since the two worlds are statistically close up until this point. If $\mathcal{Z}$ can do this with probability ϵ then the probability that $\mathcal{A}^*$ solves the SLS problem is the same.

Overall, $\mathcal{S}$ correctly simulates the protocol Π_{Legendre} as long as $u \neq 0$, which happens with probability $\leq 1/p + \epsilon$ ($s = 0$ or solving SLS with probability ϵ).

□

Perfect Correctness.

The basic protocol above is only statistically correct, as $s^2 = 0$ with probability $1/p$, and if this occurs the output will always be zero. Although this suffices for most applications, we note that perfect correctness can be obtained, at the expense of a protocol that runs in *expected* constant rounds. We can guarantee that the square s^2 is non-zero by computing it as follows:

- Take a random square $[s^2]$ and a random value $[y]$.
- Compute $[v] = [y \cdot s^2]$ and open v. If $v = 0$ then return to the first step.

Note, that the iteration of the first step only happens if $y = 0$ or $s = 0$, which occurs with probability $2/p$, so the expected number of rounds for this stage of the protocol is one.

4.4 Domain and Codomain Extension

Some applications may require a PRF which takes multiple finite field elements as input, and outputs a finite field element. We now present how to extend the basic PRF $F_{\mathsf{Leg(bit)}}$ to a function which takes messages consisting of n finite field elements and outputs a single uniformly random finite field element. Indeed our input could consist of up to t elements in the finite field where $t \leq n$. In practice we will take $n = 1$ or 2, and can then extend to larger lengths using CBC-mode or Merkle-Damgård (as in Section 1.5).

We first define a statistical security parameter $2^{-\mathsf{stat}}$, which bounds the statistical distance from uniform of the output of our PRF. We let define p' to be the nearest power of two to the prime p and set $\alpha = |p - p'|$. Then if $\alpha/p < 2^{-\mathsf{stat}}$ we set $\ell = \lceil \log_2 p \rceil$, otherwise we set $\ell = \lceil \log_2 p \rceil + \mathsf{stat}$. A standard argument will then imply that the following PRF outputs values with the correct distribution.

The key for the PRF is going to be an $\ell \times (n+1)$ matrix K of random elements in $\mathbb{F}_p$, except (for convenience) that we fix the first column to be equal to one. To apply the PRF to a vector of elements $\mathbf{x} = (x_1, \ldots, x_t)$ we "pad" $\mathbf{x}$ to a vector of $n + 1$ elements as follows $\mathbf{x}' = (x_1, \ldots, x_t, 0, \ldots, 0, t)$ and then product the matrix-vector product $\mathbf{y} = K \cdot \mathbf{x}' \in (\mathbb{F}_p)^\ell$. The output of $F_{\mathsf{Leg}(n)}$ is then given by

$$F_{\mathsf{Leg}(n)}(K, \mathbf{x}) = \left(\sum_{i=0}^{\ell-1} 2^i \cdot L_p(y_i) \right) \pmod{p}.$$

This extended PRF requires one extra round of $\ell \cdot (n-1)$ secure multiplications compared to $F_{\mathsf{Leg(bit)}}$.

Since the matrix K is compressing, the distribution of $\mathbf{y}$ will act, by the leftover hash lemma, as a random vector in $\mathbb{F}_p^\ell$. With probability ℓ/p we have $y_i \neq 0$ for all i, which implies that the values of $L_p(y_i)$ behave as uniform random bits, assuming our previous conjectures on the Legendre symbol. Thus the output value of $F_{\mathsf{Leg}(n)}(K, \mathbf{x})$ will, by choice of ℓ, have statistical distance from uniform in $\mathbb{F}_p$ bounded by $2^{-\mathsf{stat}}$.

Our choice of padding method, and the choice of the first matrix column to be equal to one, is to ensure that in the case of $n = 1$ we have

$$F_{\mathsf{Leg}(n)}(K, \mathbf{x}) = \left(\sum_{i=0}^{\ell-1} 2^i \cdot F_{\mathsf{Leg(bit)}}(k_i, y_i) \right) \pmod p.$$

In addition the padding method ensures protection against length extension attacks.

4.5 Performance

We measured performance using the prime $p = 2^{127} + 45$, which implied for $F_{\mathsf{Leg}(n)}$ we could take $\ell = 128$. Both $F_{\mathsf{Leg(bit)}}$ and $F_{\mathsf{Leg}(1)}$ obtain very low latencies (0.35ms and 1.2ms over LAN, respectively) due to the low number of rounds. For a PRF with small outputs, $F_{\mathsf{Leg(bit)}}$ achieves by far the highest throughput, with over 200000 operations per second. For full field element outputs, $F_{\mathsf{Leg}(1)}$ is around 128 times slower, but still outperforms AES in all metrics except for cleartext computation.

5. MIMC

5.1 F_{MiMC} Definition

MiMC is a comparatively simple block cipher design, where the plaintexts, the ciphertexts and the secret key are elements of $\mathbb{F}_p$ and can be seen as a simplification of the KN-cipher[40]. Its design is aimed at achieving an efficient implementation over a field $\mathbb{F}_p$ by minimizing computationally expensive field operations (e.g. multiplications or exponentiations).

Let p a prime that satisfies the condition $\gcd(3, p-1) = 1$. For a message $x \in \mathbb{F}_p$ and a secret key $k \in \mathbb{F}_p$, the encryption process of MiMC is constructed by iterating a round function r times. At round i (where $0 \le i < r$), the round function $F_i : \mathbb{F}_p \to \mathbb{F}_p$ is defined as:

$$F_i(x) = (x + k + c_i)^3,$$

where c_i are random constants in $\mathbb{F}_p$ (for simplicity $c_0 = c_r = 0$). The output of the final round is added with the key k to produce the ciphertext. Hence, the output of $F_{\mathsf{MiMC}}(x, k)$ is then given by

$$F_{\mathsf{MiMC}}(x, k) = (F_{r-1} \circ F_{r-2} \circ ... \circ F_0)(x) + k.$$

The condition on p ensures that the cubing function creates a permutation.

The number of rounds for constructing the keyed permutation is given by $r = \lceil \log_3 p \rceil$ - for prime fields of size 128 bits the number of rounds is equal to $r = 82$. This number of round r provides security against a variety of cryptanalytic techniques. In particular, due to the algebraic design principle of MiMC, the most powerful key recovery methods are the algebraic cryptanalytic attacks, as the Interpolation Attack and the GCD Attack. In the first one introduced by Jakobsen and Knudsen in [30], the attacker constructs a polynomial corresponding to the encryption function without any knowledge of the secret key. In particular, the attacker guesses the key of the final round, constructs the polynomial at round $r-1$ and checks it with one extra plaintext/ciphertext pair. In the second one, given two plaintext/ciphertext pairs (p^j, c^j) for $j = 1, 2$, the attacker constructs the polynomials $F_{\mathsf{MiMC}}(p^1, K) - c^1$ and $F_{\mathsf{MiMC}}(p^2, K) - c^2$ in the fixed but unknown key K. Since these two polynomials share $(K - k)$ as a factor (where k is the secret key), the attacker can find the value of k by computing the GCD of them.

If the attacker has access to a limited number of plaintext/ciphertext pairs only (at most $n < p$), then the number of round r can be reduced. In this case, the number of rounds is given by $r = \max\{\lceil \log_3 n \rceil, \lceil \log_3 p - 2\log_3(\log_3 p) \rceil\}$ - for prime field of size 128 bits, the number of rounds is equal to $r = 73$ if $n \le 2^{115}$, while $r = \lceil \log_3 n \rceil$ otherwise.

5.2 Computing F_{MiMC} in MPC

We consider two different approaches for computing F_{MiMC} in MPC, with a secret shared key and message. The basic approach is simplest, whilst the second variant has half the number of rounds of communication, with slightly more computation.

$\mathsf{MiMC}^{\mathsf{basic}}$**:** The naive way to evaluate F_{MiMC} requires one squaring and one multiplication for each of the r rounds. Using SPDZ, the squaring costs one opening in one round of communication, and the multiplication costs two openings in one round, giving a total of $3r$ openings in $2r$ rounds of communication.

$\mathsf{MiMC}^{\mathsf{cube}}$**:** If for each round we first compute a tuple $([r], [r^2], [r^3])$, where $r \xleftarrow{R} \mathbb{F}_p$, then given a secret-shared value $[x]$, we can open $y = x - r$ and obtain a sharing of x^3 by the computation

$$[x^3] = 3y[r^2] + 3y^2[r] + y^3 + [r^3]$$

which is linear in the secret-shared values so does not require interaction.

For a single MiMC encryption, we first compute all of the cube triples for each round, which takes just one round of communication by taking a preprocessed random square pair $([r], [r^2])$ and performing one multiplication to obtain $[r^3]$. Each round of the cipher then requires just one opening and a small amount of interaction. The total communication complexity is still $3r$ openings, but in only r rounds.

5.3 Performance

Using $r = 73$, we measured a latency of 12ms per evaluation for the simple protocol $\mathsf{MiMC}^{\mathsf{basic}}$, which halves to 6ms for the lower round variant, $\mathsf{MiMC}^{\mathsf{cube}}$. $\mathsf{MiMC}^{\mathsf{basic}}$ gives a very high throughput of over 8500 blocks per second (around 20% higher than $\mathsf{MiMC}^{\mathsf{cube}}$), and the offline cost is fairly low, at 34 blocks per second. In fact, apart from in latency, MiMC outperforms all the other PRFs we studied.

6. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the PRFs using the SPDZ multi-party computation protocol [26, 25], which provides active security against any number of corrupted parties. We focus here on the two-party setting, although the protocol easily scales to any number of parties with roughly a linear cost.

The two main metrics we use to evaluate performance are *latency* and *throughput*, both of which relate to the online phase of the SPDZ protocol. Latency measures the waiting time for a *single* PRF evaluation; the best possible latency is recorded by simply timing a large number of sequential executions of the PRF, and taking the average for one operation. In contrast, throughput is maximized by running many operations in parallel to reduce the number of rounds of communication. Of course, this comes at the expense of a higher latency, so a tradeoff must always be made depending on the precise application. In addition to latency and throughput, we present the cost of running the preprocessing phase and computing the PRF on cleartext data, for comparison.

Implementation Details: We implemented the protocols using the architecture of Keller et al. [32], which runs the online phase of SPDZ. This system automatically uses the minimum number of rounds of communication for a given program description, by merging together all independent openings. We extended the software to use the Miracl library for elliptic curve operations over the NIST P-256 curve, as required for the Naor-Reingold protocol. Note that although the SPDZ implementation supports multi-threading, all of our online phase experiments are single-threaded to simplify the comparison.

	Data type	$\mathbb{F}_p$ (ms)		$\mathbb{F}_{2^{128}}$ (ms)
		128-bit	256-bit	
LAN	Triple/Sq.	0.204	0.816	0.204
	Bit	0.204	0.816	0.00014
WAN	Triple/Sq.	4.150	16.560	4.150
	Bit	4.150	16.560	0.00285

Table 2: Time estimates for generating preprocessing data in various fields using oblivious transfer.

To estimate the cost of producing the preprocessing data (multiplication triples, random bits etc.), we used figures from the recent MASCOT protocol [31], which uses OT extensions to obtain what are currently the best reported triple generation times with active security. Although in [31], figures are only given for triple generation in a 128-bit field, we can also use these times for random square and random bit generation, since each of these can be easily obtained from one secret multiplication [21]. For the Naor-Reingold PRF, we multiplied these times by a factor of 4 to obtain estimates for a 256-bit field (instead of 128), reflecting the quadratic communication cost of the protocol. [4] The costs for all of these preprocessing data types are summarized in Table 2.

Note that LowMC only requires multiplication triples in $\mathbb{F}_2$, for which the protocol of [29] could be much faster than using $\mathbb{F}_{2^{128}}$ triples. However, we are not currently aware of an implementation of this protocol, so use the $\mathbb{F}_{2^{128}}$ times for now.

[4]The experiments in [31] showed that communication is the main bottleneck of the protocol, so this should give an accurate estimate.

Benchmarking Environment: In any application of MPC, one of the most important factors affecting performance is the capability of the network. We ran benchmarks in a standard 1Gbps LAN setting, and also a simulated WAN setting, which restricts bandwidth to 50Mbps and latency to 100ms, using the Linux `tc` tool. This models a real-world environment where the parties may be in different countries or continents. In both cases, the test machines used have Intel i7-3770 CPUs running at 3.1GHz, with 32GB of RAM.

Results: The results of our experiments in the LAN and WAN environments are shown in Tables 3 and 4, respectively. All figures are the result of taking an average of 5 experiments, each of which ran at least 1000 PRF operations. We present timings for AES and LowMC purely as a comparison metric; as explained in the introduction, these are not suitable for many MPC applications as they do not operate over a large characteristic finite field.

LowMC obtains slightly better throughput and latency than AES over a LAN, with both the vector and M4R methods achieving similar performance here. In the WAN setting, LowMC gets a very high throughput of over 300 blocks per second. This is due to the low online communication cost for multiplications in $\mathbb{F}_2$ instead of $\mathbb{F}_{2^n}$ or $\mathbb{F}_p$, and the fact that local computation is less significant in a WAN. The M4R method gets half the latency of the vector method in this scenario, since the number of rounds is halved. As discussed earlier, the preprocessing for LowMC would likely be much better than AES if implemented with the protocol of [29].

In both scenarios, the Legendre PRF gives the lowest latency, even when outputing 128-bit field elements rather than bits, due to its low round complexity. The single-bit output variant achieves by far the highest throughput of all the PRFs, so would be ideally suited to an application based on a short-output PRF, such as secure computation of the (leaky) order-revealing encryption scheme in [18]. The Legendre PRF with large outputs is useful in scenarios where low latency is very important, although the preprocessing costs are expensive compared to MiMC below. However, the high cost of the Legendre PRF "in the clear" may not make it suitable for applications in which one entity is encrypting data to/from the MPC engine

The Naor-Reingold PRF also achieves a low latency — though not as good as the Legendre PRF — but it suffers greatly when it comes to throughput. Notice that in the LAN setting, the constant rounds protocol actually performs worse than the logarithmic rounds variant in all measures, showing that here the amount of computatation and communication is more of a limiting factor than the number of rounds. Profiling suggested that over 70% of the time was spent performing EC scalar multiplications, so it seems that computation rather than communication is the bottleneck in these timings. The requirement for a 256-bit field (for 128-bit security) will be a limiting factor in many applications, as will the need to bit decompose the input, if it was previously a single field element.

The MiMC cipher seems to provide a good compromise amongst all the prime field candidates, especially as it also performs well when performed "in the clear". The "cube"

PRF	Best latency	Best throughput		Prep. (ops/s)	Cleartext (ops/s)
	(ms/op)	Batch size	ops/s		
AES	7.713	2048	530	5.097	106268670
$F_{\mathsf{LowMC}}(\mathsf{vector})$	4.302	256	591	2.562	7000
$F_{\mathsf{LowMC}}(\mathsf{M4R})$	4.148	64	475	2.565	1420
$F_{\mathsf{NR}(128)}(\mathsf{log})$	4.375	1024	370	4.787	1359
$F_{\mathsf{NR}(128)}(\mathsf{const})$	4.549	256	281	2.384	1359
$F_{\mathsf{Leg(bit)}}$	0.349	2048	202969	1225	17824464
$F_{\mathsf{Leg}(1)}$	1.218	128	1535	9.574	115591
$F_{\mathsf{MiMC}}(\mathsf{basic})$	12.007	2048	8788	33.575	189525
$F_{\mathsf{MiMC}}(\mathsf{cube})$	5.889	1024	6388	33.575	189525

Table 3: Performance of the PRFs in a LAN setting

PRF	Best latency	Best throughput		Prep. (ops/s)
	(ms/op)	Batch size	ops/s	
AES	2640	1024	31.947	0.256
$F_{\mathsf{LowMC}}(\mathsf{vector})$	1315	2048	365	0.1259
$F_{\mathsf{LowMC}}(\mathsf{M4R})$	659	2048	334	0.1261
$F_{\mathsf{NR}(128)}(\mathsf{log})$	713	1024	59.703	0.2359
$F_{\mathsf{NR}(128)}(\mathsf{const})$	478	1024	30.384	0.1175
$F_{\mathsf{Leg(bit)}}$	202	1024	2053	60.241
$F_{\mathsf{Leg}(1)}$	210	512	68.413	0.4706
$F_{\mathsf{MiMC}}(\mathsf{basic})$	7379	512	59.04	1.650
$F_{\mathsf{MiMC}}(\mathsf{cube})$	3691	512	79.66	1.650

Table 4: Performance of the PRFs in a simulated WAN setting

variant, which halves the number of rounds, effectively halves the latency compared to the naive protocol. This results in a slightly worse throughput in the LAN setting due to the higher computation costs, whereas in the WAN setting round complexity is more important. Although the latency is much higher than F_{Leg}, due to the large number of rounds, MiMC achieves the best throughput for $\mathbb{F}_p$-bit outputs, with over 6000 operations per second. In addition, the pre-processing costs of MiMC are better than that of both Legendre and the Naor-Reingold PRFs.

So in conclusion there is no single PRF which meets all the criteria we outlined at the beginning. But one would likely prefer the Legendre PRF for applications which require low latency, and which do not involve any party external to the MPC engine, and MiMC for all other applications.

7. ACKNOWLEDGEMENTS

The work in this paper has been partially supported by the ERC via Advanced Grant ERC-2010-AdG-267188-CRIPTO, the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070, and by the Austrian Science Fund via project P26494-N15.

8. REFERENCES

[1] M. Albrecht, G. Bard, and W. Hart. Algorithm 898: Efficient multiplication of dense matrices over GF(2). *ACM Transactions on Mathematical Software (TOMS)*, 37(1):9, 2010.

[2] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. Cryptology ePrint Archive, 2016. http://eprint.iacr.org/2016/492.

[3] M. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. Cryptology ePrint Archive, Report 2016/687, 2016. http://eprint.iacr.org/2016/687.

[4] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, Apr. 2015.

[5] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In P. Rudnicki, editor, *8th ACM PODC*, pages 201–209. ACM, Aug. 1989.

[6] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 535–552. Springer, Heidelberg, Aug. 2007.

[7] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.

[8] P. Bogetoft, D. L. Christensen, I. Damgård,

M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. I. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In R. Dingledine and P. Golle, editors, *FC 2009*, volume 5628 of *LNCS*, pages 325–343. Springer, Heidelberg, Feb. 2009.

[9] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 578–595. Springer, Heidelberg, Aug. 2011.

[10] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 563–594. Springer, Heidelberg, Apr. 2015.

[11] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knežević, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, Heidelberg, Dec. 2012.

[12] J. Bourgain, M. Z. Garaev, S. V. Konyagin, and I. E. Shparlinski. On the hidden shifted power problem. *SIAM Journal on Computing*, 41(6):1524–1557, 2012.

[13] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In *Fast Software Encryption - 23nd International Workshop, FSE 2016*, 2016.

[14] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for Boolean queries. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 353–373. Springer, Heidelberg, Aug. 2013.

[15] O. Catrina and S. de Hoogh. Improved primitives for secure multiparty integer computation. In J. A. Garay and R. D. Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 182–199. Springer, Heidelberg, Sept. 2010.

[16] O. Catrina and S. de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. In D. Gritzalis, B. Preneel, and M. Theoharidou, editors, *ESORICS 2010*, volume 6345 of *LNCS*, pages 134–150. Springer, Heidelberg, Sept. 2010.

[17] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In R. Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 35–50. Springer, Heidelberg, Jan. 2010.

[18] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu. Practical order-revealing encryption with limited leakage. In *Fast Software Encryption - 23nd International Workshop, FSE 2016*, 2016.

[19] J. Daemen, M. Peeters, G. Van Assche, and V. Rijmen. Nessie proposal: Noekeon. In *First Open NESSIE Workshop*, 2000.

[20] I. Damgård. On the randomness of legendre and jacobi sequences. In S. Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 163–172. Springer, Heidelberg, Aug. 1990.

[21] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In S. Halevi and T. Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 285–304. Springer, Heidelberg, Mar. 2006.

[22] I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen. Asynchronous multiparty computation: Theory and implementation. In S. Jarecki and G. Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 160–179. Springer, Heidelberg, Mar. 2009.

[23] I. Damgård and M. Keller. Secure multiparty AES. In R. Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 367–374. Springer, Heidelberg, Jan. 2010.

[24] I. Damgård, M. Keller, E. Larraia, C. Miles, and N. P. Smart. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In I. Visconti and R. D. Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 241–263. Springer, Heidelberg, Sept. 2012.

[25] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In J. Crampton, S. Jajodia, and K. Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, Sept. 2013.

[26] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, Aug. 2012.

[27] I. Damgård and R. W. Zakarias. Fast oblivious AES A dedicated application of the MiniMac protocol. In *AFRICACRYPT 2016*, pages 245–264, 2016.

[28] S. Duval, V. Lallemand, and Y. Rotella. Cryptanalysis of the FLIP family of stream ciphers. In *CRYPTO 2016*, 2016.

[29] T. K. Frederiksen, M. Keller, E. Orsini, and P. Scholl. A Unified Approach to MPC with Preprocessing Using OT. In T. Iwata and J. H. Cheon, editors, *ASIACRYPT 2015*, volume 9452 of *LNCS*, pages 711–735. Springer, Heidelberg, Dec. 2015.

[30] T. Jakobsen and L. R. Knudsen. The interpolation attack on block ciphers. In E. Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 28–40. Springer, Heidelberg, Jan. 1997.

[31] M. Keller, E. Orsini, and P. Scholl. MASCOT: Faster malicious arithmetic secure computation from oblivious transfer. Cryptology ePrint Archive, 2016. http://eprint.iacr.org/2016/505.

[32] M. Keller, P. Scholl, and N. P. Smart. An architecture for practical actively secure MPC with dishonest majority. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, pages 549–560. ACM Press, Nov. 2013.

[33] S. Laur, R. Talviste, and J. Willemson. From oblivious

AES to efficient and secure database join in the multiparty setting. In M. J. Jacobson Jr., M. E. Locasto, P. Mohassel, and R. Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 84–101. Springer, Heidelberg, June 2013.

[34] C. H. Lim and T. Korkishko. mCrypton - a lightweight block cipher for security of low-cost RFID tags and sensors. In J. Song, T. Kwon, and M. Yung, editors, *WISA 05*, volume 3786 of *LNCS*, pages 243–258. Springer, Heidelberg, Aug. 2006.

[35] Y. Lindell and B. Riva. Blazing fast 2PC in the offline/online setting with security for malicious adversaries. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 579–590. ACM Press, Oct. 2015.

[36] S. Lu and R. Ostrovsky. Distributed oblivious RAM for secure two-party computation. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 377–396. Springer, Heidelberg, Mar. 2013.

[37] P. Méaux, A. Journault, F. Standaert, and C. Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In M. Fischlin and J. Coron, editors, *EUROCRYPT 2016*, volume 9665 of *Lecture Notes in Computer Science*, pages 311–343. Springer, 2016.

[38] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, Oct. 1997.

[39] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, Aug. 2012.

[40] K. Nyberg and L. R. Knudsen. Provable security against a differential attack. *Journal of Cryptology*, 8(1):27–37, 1995.

[41] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, Dec. 2009.

[42] A. Russell and I. Shparlinski. Classical and quantum polynomial reconstruction via Legendre symbol evaluation. *Journal of Complexity*, 20(2-3):404–422, 2004.

[43] W. Van Dam. Quantum algorithms for weighing matrices and quadratic residues. *Algorithmica*, 34(4):413–428, 2002.

[44] W. van Dam, S. Hallgren, and L. Ip. Quantum algorithms for some hidden shift problems. In *14th SODA*, pages 489–498. ACM-SIAM, Jan. 2003.

[45] F. Vercauteren. The hidden root problem. In S. D. Galbraith and K. G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 89–99. Springer, Heidelberg, Sept. 2008.