View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Open Repository and Bibliography - Luxembourg

Cryptanalysis of the Legendre PRF and Generalizations*

Ward Beullens[1], Tim Beyne[1], Aleksei Udovenko[2], and Giuseppe Vitto[2]

[1] imec-COSIC, ESAT, KU Leuven, Belgium
ward.beullens@esat.kuleuven.be,tim.beyne@esat.kuleuven.be
[2] SnT, University of Luxembourg
aleksei.udovenko@uni.lu, giuseppe.vitto@uni.lu

Keywords: Cryptanalysis · Legendre PRF · MPC-friendly primitives · Collision attack

Abstract. The *Legendre PRF* relies on the conjectured pseudorandomness properties of the Legendre symbol with a hidden shift. Originally proposed as a PRG by Damgård at CRYPTO 1988, it was recently suggested as an efficient PRF for multiparty computation purposes by Grassi *et al.* at CCS 2016. Moreover, the Legendre PRF is being considered for usage in the Ethereum 2.0 blockchain.
This paper improves previous attacks on the Legendre PRF and its higher-degree variant due to Khovratovich by reducing the time complexity from $\mathcal{O}(p \log p/M)$ to $\mathcal{O}(p \log^2 p/M^2)$ Legendre symbol evaluations when $M \leq \sqrt[4]{p}$ queries are available. The practical relevance of our improved attack is demonstrated by breaking two concrete instances of the PRF proposed by the Ethereum foundation. Furthermore, we generalize our attack in a nontrivial way to the higher-degree variant of the Legendre PRF and we point out a large class of weak keys for this construction.
Lastly, we provide the first security analysis of two additional generalizations of the Legendre PRF originally proposed by Damgård in the PRG setting, namely the Jacobi PRF and the power residue PRF.

1 Introduction

The Legendre symbol is a multiplicative function modulo an odd prime number p that assigns to an element $a \in \mathbb{F}_p$ the value $1, 0$ or -1 depending on whether

* The work of Ward Beullens and Tim Beyne were supported by a PhD Fellowship from the Research Foundation - Flanders (FWO). The work of Aleksei Udovenko and Giuseppe Vitto were supported by the Luxembourg National Research Fund (FNR) project FinCrypt (C17/IS/11684537). The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg [VBCG14] – see https://hpc.uni.lu.

or not a is a square. Specifically,

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a = b^2 \text{ for some } b \in \mathbb{F}_p^\times \ , \\ 0 & \text{if } a = 0 \ , \\ -1 & \text{otherwise} \ . \end{cases}$$

The distribution of Legendre symbols has been a subject of study for number theorists at least since the early 1900s [Ala96, vS98, Jac06, Dav31, Dav39]. In particular, it follows from the Weil bound [Wei48] that the number of occurrences of a fixed pattern of l nonzero Legendre symbols among the integers $1, 2, \ldots, p-1$ modulo p is

$$\frac{p}{2^l} + \mathcal{O}(\sqrt{p}) \, ,$$

as $p \to \infty$. In other words, the distribution of fixed length substrings of Legendre symbols converges to the uniform distribution.

In 1988, Damgård [Dam90] conjectured pseudorandom properties of the sequence

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

where k has been sampled from $\mathbb{F}_p$ uniformly at random. He proposed to use this construction as a pseudorandom number generator. More recently, Grassi *et al.* [GRR+16] have proposed the same construction as a candidate pseudorandom function and have shown that it can be evaluated very efficiently in the secure multiparty computation setting. Concretely, the *Legendre pseudorandom function* $L_k(x)$ is defined by mapping the Legendre symbol with a secret shift k to $\{0, 1\}$:

$$L_k(x) = \left\lfloor \frac{1}{2}\left(1 - \left(\frac{k+x}{p}\right)\right) \right\rfloor ,$$

where p is a public prime number.

Damgård's work additionally considers several generalizations of the Legendre PRG that could be more efficient and/or more secure. One of these is to replace the Legendre symbols above by Jacobi symbols. In this case, the public modulus n is taken to be a product $\prod_i p_i$ of odd primes. Recall that the Jacobi symbol of $a \in \mathbb{F}_p$ is defined as

$$\left(\frac{a}{n}\right) = \prod_i \left(\frac{a}{p_i}\right).$$

Damgård argues that Jacobi symbols are more secure by showing that the Jacobi generator is strongly unpredictable if the Legendre generator is weakly unpredictable. Further, he notes that calculating Jacobi symbols is more efficient because computing them reduces to computing Legendre symbols modulo each of the smaller prime factors. A second generalization proposed by Damgård is the use of higher power residue symbols instead of quadratic residue symbols. Concretely, for a prime p with $p \equiv 1 \mod r$, he proposes to use the r-th power

residue symbol $a \mapsto a^{(p-1)/r} \mod p$. This potentially increases the throughput of the PRF, because we now obtain $\log_2 r$ bits of output per PRF call rather than a single bit.

Very recently, the Legendre PRF was proposed to be used in the Ethereum 2.0 proof-of-custody mechanism [Fei19b]. In this context, several cryptanalysis bounties were announced by the Ethereum foundation during the CRYPTO 2019 rump session [Fei19a]. Among the proposed challenges, there are concrete instances of the Legendre PRF with expected security levels ranging from 44 to 128 bits of security. For each instance, 2^{20} sequential output bits are given and the goal is to recover the secret key.

Despite the longevity of Damgård's pseudorandomness conjecture and the recent surge of application-oriented interest in the Legendre PRF, relatively few cryptanalytic results are available. It is known that, given quantum query access to L_k, the key k can be recovered with a single query to L_k and in quantum polynomial time [vDH00]. No subexponential attacks are known in the classical setting or the setting where a quantum adversary is only allowed to query L_k classically.

The best cryptanalytic results in the classical setting are due to Khovratovich [Kho19], who gives a memoryless birthday-bound attack. His attack recovers the key with a computational cost of $\mathcal{O}(\sqrt{p}\log p)$ Legendre symbol evaluations when given $\sqrt{p}\log p$ queries to L_k. Khovratovich also considers a higher-degree variant of the Legendre PRF, where the output is computed as the Legendre symbol of a secret polynomial in the input. Similar to the Jacobi symbol generalization, the higher-degree Legendre PRF potentially offers security and efficiency benefits.

Contributions. This paper aims to advance the state-of-the-art in the cryptanalysis of the Legendre PRF by improving upon Khovratovich's attacks on the one hand, and by providing the first security analysis of the Jacobi and power residue symbol generalizations on the other hand. Table 1 provides a summary of our main results. The practical relevance of our attacks is demonstrated by our solution of the first two concrete Legendre PRF challenges proposed by the Ethereum foundation [Fei19b]. These were expected to correspond to a security level of 44 and 54 bits, but our attacks imply that the actual security levels for these challenges are significantly lower.

After introducing the necessary preliminaries in Section 2, we show how the Khovratovich attack can be significantly improved in the low-data setting. In particular, for $M \leq \sqrt[4]{p}$ queries, the attack in Section 3 of this paper recovers the key with a time-complexity of $\mathcal{O}(p\log^2 p/M^2)$ Legendre symbol evaluations and a memory cost of $\mathcal{O}(M^2)$. In Section 4, the attack from Section 3 is generalized to the higher-degree case. As before, this amounts to a significant improvement in the low-data setting. In addition, for $d \geq 3$ and with $M = p$ queries, we gain a factor of p in time-complexity compared to Khovratovich's results. Furthermore, in Section 4, a large class of weak keys for the higher-degree Leg-

endre PRF is shown to exist. For keys in this class, key-recovery requires roughly $\mathcal{O}(p^{\lfloor d/2 \rfloor} d \log p)$ Legendre symbol evaluations with only $d\lceil \log p \rceil$ queries to the PRF. This attack requires $\mathcal{O}(p^{\lceil d/2 \rceil} d \log p)$ memory, but trade-offs are available using Van Oorschot-Wiener golden collision search. We also give a reduction to the unique k-XOR problem, which results in further time-memory trade-offs.

The first of Damgård's generalizations is discussed in Section 6. Specifically, it will be shown that the Jacobi PRF can be broken with cost proportional to the cost of breaking the Legendre PRF for each of the prime factors of the modulus separately. The power residue symbol generalization is analyzed in Section 7. Besides the straightforward generalization of the attack from Section 3 to the r-th power residue symbol PRF, we additionally provide a more efficient attack for the case where r is large.

Finally, concrete implementation results are provided in Section 8. We report on the specific amount of time and memory that was necessary to solve the first two Legendre PRF challenges of the Ethereum foundation. These results showcase the practical relevance of our attacks.

Table 1. Query, time and memory requirements of previous and new attacks on the Legendre PRF. The reported time and memory values are asymptotic upper bounds ($\mathcal{O}$-notation), ℓ and s denote the time-complexity of computing a Legendre and power residue symbol respectively. The attack strategy for composite moduli from Section 6 can be combined with any of the attacks in this table.

	Reference	Queries	Time	Memory
Legendre PRF	Randomized [Kho19]	$\log p$	$\ell p \log p$	$\log p$
	Khovratovich [Kho19]	$\sqrt{p} \log p$	$\ell \sqrt{p} \log p$	$\log p$
	Section 3.1	M	$M + \ell p \log p / M$	$M \log p$
	Section 3.3	M	$M^2 + \ell p \log^2 p / M^2$	M^2
	Section 3.4	M	$M^2 + p \log^2 p / M^2$	$M^2 / \log p$
Degree $d \geq 2$ Legendre PRF	Randomized [Kho19]	$\log p$	$\ell p^d \, d \log p$	$d \log p$
	Khovratovich [Kho19]	p	$\ell p^{d-1} d \log p$	$d \log p$
	Section 4	M	$M^2 + \ell p^d d^2 \log^2 p / M^2$	M^2
	Section 5	$d \log p$	$\ell p^{\lfloor d/2 \rfloor} d \log p$	$p^{\lceil d/2 \rceil} d \log p$
r-th power-residue PRF	Section 7.2	M	$M^2 + sp \log^2 p / (M^2 \log^2 r)$	$M^2 \log r$
	Section 7.3	M	$M + sp \log^2 p / (Mr \log^2 r)$	$M \log r$

2 Preliminaries

After introducing the Legendre PRF and some related notation in Section 2.1, Section 2.2 recalls how Legendre and power residue symbols can be computed

efficiently. Finally, Sections 2.3 and 2.4 discuss Khovratovich's attacks on the Legendre PRF and its higher degree variant.

2.1 Legendre PRF

Definition 1 (Legendre function). *For a given odd prime p, we consider the function*

$$\begin{aligned} l : \mathbb{F}_p &\to \mathbb{F}_2 \\ x &\mapsto \left\lfloor \frac{1}{2}\left(1 - \left(\frac{x}{p}\right)\right)\right\rfloor \end{aligned}$$

which maps quadratic residues modulo p to $0 \in \mathbb{F}_2$ and quadratic non-residues to $1 \in \mathbb{F}_2$.

Definition 2 (Legendre PRF). *Let p be an odd prime and d a positive integer. The* degree d-Legendre PRF *over $\mathbb{F}_p$ is a family of functions $L_k : \mathbb{F}_p \to \mathbb{F}_2$ such that for each $k \in \mathbb{F}_p^d$,*

$$L_k(x) = l\big(x^d + \textstyle\sum_{i=0}^{d-1} k_{i+1}\, x^i\big).$$

Remark 1. For any given field $\mathbb{F}_p$, the Legendre symbol is *multiplicative*, i.e.

$$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right) \quad \text{for all } a, b \in \mathbb{F}_p.$$

In terms of the Legendre function l, multiplication of inputs corresponds to addition in $\mathbb{F}_2$ of the respective images. Indeed

$$l(ab) = l(a) \oplus l(b) \quad \text{for all } a, b \in \mathbb{F}_p^\times,$$

where $\oplus$ denotes addition in $\mathbb{F}_2$.

In our analysis, we will often consider sequential evaluations of a given degree d Legendre PRF L_k starting from a point a with an additive or multiplicative step b. We call such vectors *L-sequences*.

Definition 3 (L-sequences). *Let p be an odd prime, m a positive integer and $a, b \in \mathbb{F}_p$. For a given L_k over $\mathbb{F}_p$, we define the* arithmetic L-sequence of length m with starting point a and stride b *as the $\mathbb{F}_2^m$-vector*

$$L_k(a + b\,[m]) := (\, L_k(a), L_k(a+b), \ldots, L_k(a + (m-1)b)\,).$$

Similarly, we define the geometric L-sequence of length m with starting point a and common ratio b *as the $\mathbb{F}_2^m$-vector*

$$L_k(a \cdot b^{[m]}) := (L_k(a),\, L_k(a \cdot b),\, \ldots,\, L_k(a \cdot b^{m-1}\,)).$$

To justify the correctness of our attack, the following property of L_k will be assumed.

Assumption 1 *Let p be an odd prime and d a positive integer. Let $m = d\lceil \log p \rceil$. For all $k \in \mathbb{F}_p^d$, there exist at most $\mathcal{O}(1)$ keys $k' \in \mathbb{F}_p^d$ such that $L_{k'}([m]) = L_k([m])$.*

2.2 Evaluating Legendre and Power Residue Symbols

Using the law of quadratic reciprocity, i.e. for odd coprime integers p and q

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}},$$

Legendre symbols (and more generally Jacobi symbols) can be computed at essentially the same cost as a GCD computation. Using the Euclidean algorithm, the cost of a Legendre symbol computation is $\mathcal{O}(\log p)$ arithmetic operations, or $\mathcal{O}(\log^2 p \log\log p)$ bit operations. Brent and Zimmerman [BZ10] give an asymptotically better algorithm with complexity $\mathcal{O}(\log p \log^2 \log p)$. Power residue symbols can be computed via modular exponentiation in time $\mathcal{O}(\log p \log(p/r) \log\log p)$. In the remainder of this paper, we will often refer to the cost of an algorithm in terms of the number of Legendre symbol computations or power residue symbol computations.

2.3 Attacks on the Linear Legendre PRF

Khovratovich [Kho19] describes a chosen plaintext attack for the linear Legendre PRF L_k that recovers $k \in \mathbb{F}_p$ with $\mathcal{O}(\sqrt{p}\log p)$ queries to L_k. The attack is based on a memoryless collision search between two specific functions and can be briefly summarized as follows.

Let $m = \lceil \log p \rceil$ and consider the functions $x \mapsto L_k(x+[m])$ and $x \mapsto L_0(x+[m])$. Note that the L-sequence $L_k(x+[m])$ is available by querying the Legendre PRF, whereas $L_0(x+[m])$ does not depend on k. By Assumption 1, a collision between $x \mapsto L_k(x+[m])$ and $x \mapsto L_0(x+[m])$ yields k with high probability. Indeed, let $a, b \in \mathbb{F}_p$ be such that $L_k(a+[m]) = L_0(b+[m])$. We have

$$L_0(a+k+[m]) = L_0(b+[m]).$$

In accordance with Assumption 1, the number of superfluous candidates for k satisfying the above equality is expected to be at most $\mathcal{O}(1)$.

Collisions between $x \mapsto L_k(x+[m])$ and $x \mapsto L_0(x+[m])$ can be found with a generic memoryless collision search method [MOM92] in $\mathcal{O}(\sqrt{p})$ evaluations of both functions. Since computing each L-sequence requires $m = \mathcal{O}(\log p)$ calls to

L_k, the overall complexity sums up to $\mathcal{O}(\sqrt{p}\log p)$ queries to L_k and L_0. More generally, if only M queries to L_k are allowed, a collision can be found with $\mathcal{O}(p\log^2 p/M)$ queries to L_0. This will be discussed in detail in Section 3.1.

We note that Khovratovich's original attack builds sequences of length m using arbitrary evaluations of the Legendre function L_k rather than consecutive ones. This difference does not affect the overall attack complexity, but by using L-sequences we will be able to reduce the data complexity in Section 3.

2.4 Attacks on the Higher-Degree Legendre PRF

Khovratovich [Kho19] also presents a generalization of the chosen plaintext attack from Section 2.3 to the quadratic case and, ultimately, to arbitrary degrees.

Let $k = (k_1, k_2) \in \mathbb{F}_p^2$ and consider the associated quadratic Legendre PRF L_k. Choose any $r \in \mathbb{F}_p^\times$. From the multiplicative property of the Legendre symbol we get that for any $a \in \mathbb{F}_p$ and $j \in \mathbb{Z}$,

$$L_{(r^{2j}\,k_1, r^j\,k_2)}(a) = l(r^{2j}) \oplus L_{(k_1,k_2)}(ar^{-j}) = L_k(ar^{-j}),^3$$

since r^{2j} is clearly a quadratic residue modulo p. Let $m = 2\lceil \log p \rceil$. If we find a $k' \in \mathbb{F}_p^2$ and a $j \in \mathbb{Z}$ such that

$$L_{k'}(r \cdot r^{[m]}) = L_k(r^{1-j} \cdot r^{[m]}),$$

then we successfully recover k by letting $k_1 = k_1' r^{-2j}$ and $k_2 = k_2' r^{-j}$. As for the linear case, such a collision can be found memorylessly with $\mathcal{O}(p)$ queries to L_k and $\mathcal{O}(p)$ Legendre symbol computations.

For the general case, consider the degree-d Legendre PRF L_k. Similarly to the quadratic case, we have for each $a \in \mathbb{F}_p$ and $j \in \mathbb{Z}$ that

$$L_{k_1 r^{dj}, k_2 r^{(d-1)j}, \ldots, k_d r^j}(a) = l(r^{dj}) \oplus L_k(ar^{-j}).$$

By guessing the coefficients $k_3, \ldots, k_d$, it is possible to attack the remaining coefficients k_1 and k_2 using geometric L-sequences of length $d\lceil \log p \rceil$ similar to the quadratic case. It follows that k can be recovered using $\mathcal{O}(p^{d-2} \cdot p \cdot d \log p) = \mathcal{O}(p^{d-1} d \log p)$ Legendre symbol evaluations, given $\mathcal{O}(p)$ queries to L_k.

3 Improved Attack on the Linear Legendre PRF

In this section, we show how Khovratovich's attack (Section 2.3) on the Legendre PRF can be improved when the total number of available queries is less than

[3] This equation, and many other equations in this paper, only holds if none of the involved Legendre symbols evaluate to zero. Since this does not pose a problem in practice we choose to ignore this issue for notational convenience.

$\sqrt{p}$. Although, in its simplest form, our method requires additional memory, we discuss several techniques to reduce memory requirements while keeping the same overall time complexity.

3.1 Table-Based Collision Search

We first transform the attack by Khovratovich into a table-based collision search.

Let M be the allowed number of queries to the oracle L_k, where $\log p \ll M < \sqrt{p}$. Let $m = \lceil \log p \rceil$ and let $\tilde{M} = M - m + 1$. The attack proceeds as follows:

1. Store in a table $\mathcal{T}$ the pairs $(L_k(a + [m]), a)$ for all $a \in \left\{0, \ldots, \tilde{M} - 1\right\}$.
2. Sample b uniformly at random from $\mathbb{F}_p$ until $(L_0(b + [m]), a) \in \mathcal{T}$ for some $a \in \{0, \ldots, \tilde{M} - 1\}$. For each a corresponding to such a collision, a candidate key $\tilde{k}$ is recovered as $\tilde{k} = b - a$. By Assumption 1, the number of candidate keys is at most $\mathcal{O}(1)$. Candidate keys $\tilde{k}$ can be tested by comparing one or more entries of $\mathcal{T}$ with the corresponding arithmetic L-sequences with starting point $\tilde{k}$.

Regarding the time and memory complexity of this attack, we note that the first step requires M queries to L_k, from which we obtain $\tilde{M}$ arithmetic L-sequences that are stored using $\mathcal{O}(M \log p)$ memory. The second step requires $\mathcal{O}(p \log p/M)$ evaluations of the Legendre symbol and no additional memory is needed. Hence, the overall computational cost of the attack is $\mathcal{O}(M + p \log p/M)$.

Note that this variant of the attack already reduces the query and time complexities by a $\log p$ factor compared to the memoryless collision search, although a significant amount of memory is employed.

Remark 2. The above attack can be made deterministic by choosing $b \in \{0, \ldots, \lfloor p/\tilde{M} \rfloor\}$ and considering the sequences $v = L_0(b\tilde{M} + [m])$ in the second step of the attack. Indeed, it is easy to see that for any $k \in \mathbb{F}_p$, the arithmetic L-sequence at offset $\tilde{M} \left\lceil k/\tilde{M} \right\rceil$ will be computed in both steps of the attack and the correct key is guaranteed to be recovered after at most $\mathcal{O}(M + p \log p/M)$ Legendre symbol evaluations.

3.2 Expanding the Number of L-Sequences

We now show that the table can be expanded without increasing the number of queries M. The key idea is to exploit the *multiplicative* property of the Legendre symbol.

Lemma 1. *Let m be a positive integer and $k \in \mathbb{F}_p$. For any $b \in \mathbb{F}_p^\times$ and $a \in \mathbb{F}_p$ it holds that*

$$L_{k/b}(a/b + [m]) = (l(b), \ldots, l(b)) \oplus L_k(a + b[m]).$$

Proof. Immediate by the multiplicative property of l.

Lemma 2. *Let $k \in \mathbb{F}_p$ and $m \leq M$ positive integers. Then from the arithmetic L-sequence $L_k([M])$, it is possible to extract $\sim M^2/m$ arithmetic L-sequences of the form $L_{k/b}(a/b + [m])$ for distinct pairs $(a, b) \in \mathbb{F}_p \times \mathbb{F}_p^\times$.*

Proof. Let b a positive integer such that $b \leq \lfloor M/m \rfloor$. By Lemma 1, we get

$$L_k(a + b[m]) = (l(b), \ldots, l(b)) \oplus L_{k/b}(a/b + [m])$$

for any $a \in [0, M - bm + 1)$, thus each b yields a total of $M - bm + 1$ L-sequences of length m. Moreover, since $L_k(a - b[m])$ is equal to the sequence $L_k(a - b(m-1) + b[m]) = L_k(a' + b[m])$ written in reverse order, we can consider negative values for b too, thus doubling the total number of sequences. Hence, the total number of arithmetic L-sequences of length m that can be extracted from $L_k([M])$ equals

$$2 \sum_{b=1}^{\lfloor M/m \rfloor} (M - bm + 1) \sim \frac{2M^2}{m} - m \sum_{b=1}^{M/m} b \sim \frac{2M^2}{m} - \frac{M^2}{m} = \frac{M^2}{m}. \square$$

3.3 An Improved Table-Based Collision Search

The observations from Section 3.2 will now be used to improve the table-based collision search from Section 3.1.

As before, let M be the allowed number of queries to the oracle L_k, where $\log p \ll M < \sqrt{p}$. Let $m = \lceil \log p \rceil$. The attack proceeds as follows:

1. Query the sequence $L_k([M])$ and extract $\sim M^2/m$ sequences of the form $L_{k/b}(a/b + [m])$ from it. This is possible by Lemma 2. Store all of the triples $(L_{k/b}(a/b + [m]), a, b)$ in a table $\mathcal{T}$.
2. Sample c uniformly at random from $\mathbb{F}_p$ until $(L_0(c + [m]), a, b) \in \mathcal{T}$ for some a and b. For each pair (a, b) corresponding to such a collision, a candidate key $\tilde{k}$ is recovered as $\tilde{k} = bc - a$. By Assumption 1, the number of candidate keys is at most $\mathcal{O}(1)$. As before, the correctness of candidate keys $\tilde{k}$ can easily be verified.

The first step of the attack requires M queries to L_k and $\sim M/m$ Legendre symbol evaluations. Storing the table $\mathcal{T}$ requires $\mathcal{O}(M^2)$ memory. In the second

phase, an average of $\sim mp/M^2$ samples must be tested before a collision is found. Hence, the computational cost of this step is dominated by $\mathcal{O}(pm^2/M^2)$ Legendre symbol evaluations.

It follows that the overall cost of the attack is dominated by the extraction of $\mathcal{O}(M^2/m)$ sequences, the evaluation of $\mathcal{O}(M/m+p\log^2 p/M^2)$ Legendre symbols and a memory requirement of $\mathcal{O}(M^2)$. For $M < \sqrt{p}$, this is always an improvement over the attack from Section 3.1 – possibly after discarding some of the data.

3.4 Additional Optimizations

This section describes a number of additional optimizations that allow a further reduction of both the time and the memory complexity of the attack by a factor $\Omega(\log p)$.

Using Consecutive Values of c The second step of the attack from Section 3.3 can be optimized by choosing consecutive values of c rather than uniform random samples. This approach allows us to reuse most of the Legendre symbol computations since, for example, $L_0(c + [m])$ and $L_0(c + 1 + [m])$ overlap almost completely. A priori, this allows reducing the number of Legendre symbol computations by a factor of $\Omega(m)$. However, there is an important caveat: since the guesses for c are not independent, the expected number of iterations of the second step is no longer pm/M^2. To see why this is the case, recall that for any c, the algorithm will output the correct key k if there exists $(*, a, b) \in \mathcal{T}$ such that $k = bc - a$. Since the table contains an entry $(*, a, b)$ for all sufficiently small values of a and b, it is clear that if the table contains $(*, a, b)$ such that $k = bc - a$ it is likely to also contain $(*, a' = a + b, b)$ such that $k = b(c + 1) - a'$. Therefore, if c is a good guess, then $c + 1$ is also likely to be a good guess. Since the "good" values of c are clustered together in groups of size $\mathcal{O}(m)$, we expect the required number of iterations to be $\mathcal{O}(pm^2/M^2)$, which means that the factor $\Omega(m)$ that we saved by using consecutive guesses for c is lost again. However, we can still use this idea to reduce the memory complexity of the algorithm: by only storing one entry $(*, a, b)$ for each cluster of good c's, i.e. we only store the triples $(*, a, b)$ such that $|a| < |b|$, the size of the table can be reduced by a factor of $\Omega(m)$ without impacting the time complexity of the attack.

Expanding the Number of L-Sequences in the Second Step The idea outlined in Section 3.2 can be used to create new L-sequences from those computed during the second step of the attack. Indeed, after computing a large number of $w = \Omega(m)$ consecutive Legendre symbols $L_0(c + [w])$, it is possible to extract $\Omega(w^2/m^2)$ arithmetic subsequences of the form $L_0(c + c' + d[m])$ such that $|c'| < |d|$, with no need to compute additional Legendre symbols. Using the

property that

$$L_0(c + c' + d[m]) = L_0((c + c')/d + [m]) \oplus L_0(d)$$

we can then do $\Omega(w^2/m^2)$ table lookups. Asymptotically, this allows to amortize away the cost of computing Legendre symbols, so the time complexity is dominated by the extraction of $\mathcal{O}(pm^2/M^2)$ subsequences rather than by the computation of $\mathcal{O}(pm^2/M^2)$ Legendre symbols.

Not Storing Reverse Sequences Since the sequence $a+b[m]$ is just the reverse of the sequence $a + b(m-1) - b[m]$, there is some redundancy in the lookup table. Indeed, for each entry $(s, a, b) \in \mathcal{T}$, the reverse sequence corresponding to the entry $(s', a+b(m-1), -b)$ is also stored. If, instead, we only store either the sequence or its reverse (e.g. by storing the lexicographically smallest sequence), then the memory requirements are reduced by a factor of two without affecting the overall time-complexity just by looking up either the sequence $L_0(c + [m])$ *or* its reverse in $\mathcal{T}$, depending which comes first lexicographically.

4 Application to the Higher-Degree Legendre PRF

In this section we generalize the attack described in Section 3 to Legendre PRFs of degree $d > 1$. In Section 4.1 it is shown how to expand the number of L-sequences in the higher-degree setting. The resulting attack is detailed in Section 4.2.

4.1 Expanding the Number of L-Sequences

In order to generalize Lemma 2, we need to extend Lemma 1 to the higher-degree case. This is the object of Lemma 3.

Lemma 3. *For any positive integer m, $b \in \mathbb{F}_p^\times$ and $a \in \mathbb{F}_p$, there exists an invertible affine transformation $T_{a,b}$ such that for any $k \in \mathbb{F}_p^d$,*

$$L_{T_{a,b}(k)}([m]) = (l(b^d), \ldots, l(b^d)) \oplus L_k(a + b[m]).$$

Moreover, for any choice of $(a, b) \in \mathbb{F}_p \times \mathbb{F}_p^\times$, the transformation $T_{a,b}$ can be efficiently computed.

Proof. Lef f be the monic degree d polynomial with coefficient vector k, and let $T_{a,b}(k)$ be the coefficient vector of the monic polynomial $f(a+bx)/b^d$. Then, by the multiplicative property of the Legendre symbol, we have that

$$L_{T_{a,b}(k)}([m]) = (l(b^d), \ldots, l(b^d)) \oplus L_k(a + b[m]).$$

Furthermore, it is not hard to see that $T_{a,b}$ is invertible, affine and that it can be computed efficiently.

Lemma 4. *Let $k \in \mathbb{F}_p^d$ and $m \leq M$ positive integers. Then from the arithmetic L-sequence $L_k([M])$, it is possible to extract $\sim M^2/m$ arithmetic L-sequences of the form $L_{k'}([m])$ with k' as defined in Lemma 3 for distinct pairs $(a, b) \in \mathbb{F}_p \times \mathbb{F}_p^\times$.*

Proof. The proof is completely analogous to that of Lemma 2.

4.2 An Improved Table-Based Collision Search

The attack proceeds in essentially the same way as described in Section 3.3 for the linear case. Let M be the allowed number of consecutive queries to the oracle L_k. Let $m = d\lceil \log p \rceil$. The attack comprises the following steps:

1. Query the sequence $L_k([M])$ and extract $\sim M^2/m$ sequences of the form $L_{k'}([m])$ from it. This is possible by Lemma 4. Store all of the triples $(L_{k'}([m]), a, b)$ in a table $\mathcal{T}$.
2. Sample k' uniformly at random from $\mathbb{F}_p^d$ until $(L_{k'}([m]), a, b) \in \mathcal{T}$ for some a and b. For each pair (a, b) corresponding to such a collision, a candidate key $\tilde{k}$ can be recovered from k, a and b as in Lemma 3. By Assumption 1, the number of candidate keys is at most $\mathcal{O}(1)$. As before, the correctness of candidate keys can easily be verified.

As in Section 3.3, the computational cost of the first step is dominated by the extraction of $\mathcal{O}(M^2/m)$ sequences. For the second step, at most $\mathcal{O}(p^d\, m^2/M^2)$ Legendre symbols are expected to be evaluated. Hence, the total computational cost of the attack consists of $\mathcal{O}(M^2/m)$ sequence extractions and $\mathcal{O}(p^d\, d^2 \log^2 p/M^2)$ Legendre symbol evaluations. The attack requires $\mathcal{O}(M^2)$ memory.

For $d \geq 3$, the time-complexity is minimized for $M = p$. The time complexity is then $\mathcal{O}(p^{d-2} d^2 \log^2 p)$ Legendre symbol computations. Hence, we gain a factor of p in time relative to the attacks by Khovratovich [Kho19].

5 Weak Keys in the Higher-Degree Legendre PRF

In this section, we exhibit a large class of weak keys for the higher-degree Legendre PRF. Our attacks are based on the observation that for some keys, the corresponding monic polynomial factors as a product of polynomials of lower degree.

5.1 A Birthday-Bound Attack for Some Keys

Consider the Legendre PRF of degree $d \geq 2$ over $\mathbb{F}_p$ for a prime p. Recall that the key $k \in \mathbb{F}_p^d$ of the PRF corresponds to the monic polynomial $f(x) =$

$x^d + \sum_{i=0}^{d-1} k_{i+1} x^i \in \mathbb{F}_p[x]$. The attack in this section is based on the observation that, with high probability, the polynomial f has a factor of degree $t = \lfloor d/2 \rfloor$. In this case, there exist two monic polynomials $g, h \in \mathbb{F}_p[x]$ with $\deg g = t$ and $\deg h = d - t$ such that $f = gh$.

Assume that we are given the outputs of the PRF on $m = d\lceil \log p \rceil$ arbitrary inputs, for example the sequence $L_k([m])$. Then, by the multiplicative property of the Legendre symbol[4],

$$L_k([m]) = l(g([m])) \oplus l(h([m])).$$

Hence, the problem of finding the secret key $k \in \mathbb{F}_p^d$ reduces to a simple collision search:

1. Query the sequence $L_k([m])$ from the PRF. For each monic polynomial g of degree t, store the pair $(L_k([m]) \oplus l(g([m])), g)$ in a table $\mathcal{T}$.
2. Sample monic polynomials h of degree $d-t$ until $(l(f([m])), g) \in \mathcal{T}$ for some monic polynomial g of degree t. For each such g, recover a candidate key from the coefficients of gh. By Assumption 1, the number of candidate keys will be at most $\mathcal{O}(1)$.

For $t = \lceil d/2 \rceil$, this attack requires $\mathcal{O}(p^{\lceil d/2 \rceil} d \log p)$ memory and its time complexity is dominated by $\mathcal{O}(p^{\lfloor d/2 \rfloor} d \log p)$ Legendre symbol computations. The attack requires only $m = \mathcal{O}(d \log p)$ queries to the PRF.

Using Van Oorschot-Wiener golden collision search [vOW94], an improved time-memory trade-off can be obtained: given M bits of memory, the key can be recovered with a time-complexity of $\mathcal{O}(d \log p \sqrt{p^{3d/2}/M})$ Legendre symbol evaluations.

Even if the polynomial f does not have a factor of degree exactly $\lceil d/2 \rceil$, it might still have a factor of large degree $t < \lceil d/2 \rceil$. In this case, the same strategy results in an attack with time complexity $\mathcal{O}(p^{d-t} d \log p)$ and memory complexity $\mathcal{O}(p^t d \log p)$. This gives a trade-off between more efficient attacks on a smaller fraction of keys (when t is large) or less efficient attacks on a larger fraction of the keys (when t is small). This trade-off is illustrated in Figure 1. The figure shows the time-complexity of the attack for a desired fraction of attackable keys. The construction of Figure 1 is based on the following fact [Tao15]: the fraction of monic degree-d polynomials whose factorization has exactly c_i monic irreducible factors of degree i is $1/\prod_{i=1}^{d} c_i!\, i^{c_i}$ as $p \to \infty$. By summing these probabilities over all integer partitions of d that allow a $(t, d-t)$ split, we obtain the probability that a uniformly random key is weak.

We conclude that if the key is chosen uniformly at random, the higher-degree Legendre PRF has security only up to the birthday bound. To completely prevent this class of attacks, one can choose the key k such that the corresponding polynomial f is irreducible.

[4] For convenience, we extend our notation for arithmetic L-sequences (Definition 3) to arbitrary functions on $\mathbb{F}_p$. In particular, $l(g([m])) = (l(g(0)), \ldots, l(g(m-1)))$.

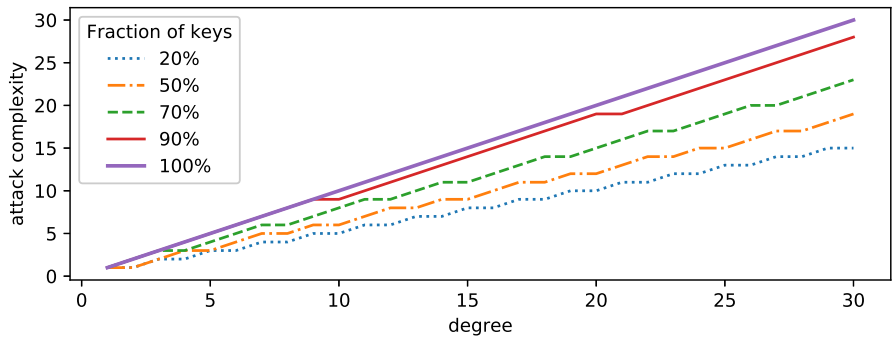

Fig. 1. The complexity of the attack, measured as a power of p, as a function of the degree of f and the desired fraction of keys we want to attack.

5.2 Reduction to the Unique k-XOR Problem

More generally, the secret polynomial could factor into k polynomials of degree roughly d/k. For example, if d is divisible by k and $f = \prod_{i=1}^{k} f_i$ with $\deg f_i = d/k$, we have

$$L_k([m]) = \bigoplus_{i=1}^{k} l(f_i([m])).$$

That is, it suffices to find a solution to a variant of the k-XOR problem. Specifically, since each list has length $p^{d/k}$, a unique solution is expected. This makes Wagner's approach [Wag02] inapplicable, but some improvements over the attack in Section 5.1 are nevertheless possible.

In particular, for $k = 4$, the algorithm of Chose, Joux and Mitton [CJM02] leads to a time complexity $\tilde{\mathcal{O}}(p^{d/2})$ with only $\tilde{\mathcal{O}}(p^{d/4})$ memory. Corresponding time-memory trade-offs can also be obtained.

Finally, we mention that there exist asymptotically better quantum algorithms for the unique k-XOR problem. Bernstein *et al.* [BJLM13] give an $\widetilde{\mathcal{O}}(p^{0.3d})$ algorithm requiring $\widetilde{\mathcal{O}}(p^{0.2n})$ quantum-accessible quantum memory for $k = 4$. For any $k \geq 3$, Naya-Plasencia and Schrottenloher [NPS19] give algorithms running in time $\widetilde{\mathcal{O}}(p^{\beta_k d})$ where $\beta_k = (k + \lceil k/5 \rceil)/(4k)$ using $\widetilde{\mathcal{O}}(p^{0.2n})$ quantum-accessible quantum memory. For $k = 3$, there is an algorithm using $\widetilde{\mathcal{O}}(p^{d/3})$ time and $\widetilde{\mathcal{O}}(p^{d/3})$ quantum-accessible *classical* memory.

6 Jacobi Symbol PRF

The Jacobi pseudorandom generator was proposed by Damgård [Dam90] as a variation on the Legendre PRG. As discussed by Damgård [Dam90, §5], it is po-

tentially more efficient because it can be computed as the exclusive-or of several Legendre PRGs with a relatively small modulus. In addition, Damgård showed that if the Legendre generator is weakly unpredictable, then the Jacobi generator is strongly unpredictable. A generator is defined to be weakly unpredictable if, for all polynomials f, there exist only finitely many integers $m \geq 0$ such that the next output bit in a sequence of length m can be predicted with probability greater than $1 - 1/f(m)$. Similarly, the generator is said to be strongly unpredictable if the probability of successful prediction exceeds $1/2 + 1/f(m)$ for only finitely many m. For a more formal definition, see [Dam90, §3] and references therein.

This section investigates the security of the Jacobi PRF in the chosen-plaintext setting. Whereas the unpredictability result of Damgård could be regarded as a positive result related to the security of the Jacobi PRF, it remains inconclusive concerning its concrete security. Indeed, strong unpredictability is a weaker property than PRF-security and, in addition, it is only an asymptotic notion of security.

Clearly, the cost of attacking the Jacobi PRF is at least the cost of attacking a Legendre PRF corresponding to a prime factor of the modulus. Below, a chosen-plaintext key-recovery attack on the Jacobi PRF is given which nearly attains this lower bound. Hence, for most purposes, the Jacobi PRF offers little benefit over the Legendre PRF.

Let $n = \prod_{i=1}^{m} p_i$ with $p_1, \ldots, p_m$ distinct odd primes. Note that it may be assumed that the prime factors of n are distinct, since

$$\left(\frac{x+k}{n}\right) = \left(\frac{x+k}{\prod_{i=1}^{m} p_i^{e_i}}\right) = \prod_{\substack{i=1 \\ e_i \text{ odd}}}^{m} \left(\frac{x+k}{p_i}\right).$$

Let $\lambda_j = \prod_{\substack{i=1 \\ i \neq j}}^{m} p_i$ and denote the inverse of λ_j modulo p_j by λ'_j. Then

$$\left(\frac{\lambda_j\, x + k}{n}\right) = \prod_{i=1}^{m} \left(\frac{\lambda_j\, x + k}{p_i}\right) = \left(\frac{\lambda_j}{p_j}\right) \left(\frac{k}{n/p_j}\right) \left(\frac{x + \lambda'_j\, k}{p_j}\right).$$

Hence, in the chosen-plaintext setting, the key-recovery attack on the Legendre PRF from Section 3 can be used to recover the key modulo p_j. The factor $\left(\frac{k}{n/p_j}\right)$ is not known to the attacker, but it is constant so the cost of the attack is increased by a factor of at most two. Given the value of the key modulo each prime factor of n, the Chinese remainder theorem yields the value of the key modulo n. Hence, key recovery for the Jacobi symbol costs at most $\mathcal{O}(mM^2 + \sum_{i=1}^{m} p_i \log^2 p_i / M^2)$ Legendre symbol evaluations. The same strategy is applicable to the higher-degree case and can also be combined with the attacks in Section 7 below.

7 Attacks on the Power Residue PRF

The MPC protocol of Grassi *et al.* [GRR+16] for computing the Legendre PRF requires only three rounds of communication, which makes the Legendre PRF superior among the PRF constructions investigated by Grassi *et al.* in terms of latency. However, since the Legendre PRF only produces one bit of output, it compares less favorably in terms of throughput than e.g. MiMC [AGR+16], a block cipher that outputs full field elements.

To mitigate this limitation of the Legendre PRF we can, as proposed by Damgård [Dam90], consider higher power residue symbols rather than quadratic residue symbols. If r divides $p-1$, the *r-th power residue symbol* of $x \in \mathbb{F}_p$ is defined as

$$\left(\frac{x}{p}\right)_r := x^{\frac{p-1}{r}} \mod p.$$

Jointly computing r-th power residue symbols in the MPC setting can be done at essentially the same cost as computing Legendre symbols with the advantage that $\log r$ bit outputs are produced instead. Therefore, this modification has the potential to significantly increase the throughput of the Legendre PRF at essentially no cost – keeping in mind that r should not be too large, since the corresponding power residue PRF might lose its security (e.g. $r = p-1$). In this section we provide the first security analysis of the power residue PRF. We show that there exists an attack with time complexity $\mathcal{O}(p \log^2 p/(Mr \log^2 r))$, given $M \leq \sqrt{p}$ queries to the PRF.

7.1 Power Residue PRF

By generalising the Legendre function and the Legendre PRF to higher power residues, we obtain the following definitions:

Definition 4 (r-th power residue function). *Let p be a prime congruent to 1 mod r and g a generator of $\mathbb{F}_p^\times$. Then we define the r-th power residue function $l^{(r)} : \mathbb{F}_p \to \mathbb{Z}_r$ as*

$$l^{(r)}(a) = \begin{cases} k \text{ if } a \not\equiv 0 \mod p \text{ and } a/g^k \text{ is an } r\text{-th power mod } p \\ 0 \text{ if } a \equiv 0 \mod p \end{cases}$$

Definition 5 (r-th power residue PRF). *Let p be a prime congruent to 1 modulo r. The* power residue PRF *over $\mathbb{F}_p$ is a family of functions $L_k^{(r)} : \mathbb{F}_p \to \mathbb{Z}_r$ such that for each $k \in \mathbb{F}_p$,*

$$L_k^{(r)}(x) = l^{(r)}(k + x).$$

7.2 Generalising our Attack to the Power Residue PRF

The attacks described in Section 3 and Section 4 do not use any properties of the Legendre symbol other than its multiplicativity. Therefore, they trivially generalize to any multiplicative function with a hidden shift, including the r-th power residue function.

Unlike the quadratic case, the r-th power residue function can take r distinct values, so it suffices to consider L-sequences of length $\log p / \log r$. It follows that a straightforward generalization of our attack to r-th power residue Legendre PRFs requires $\mathcal{O}(p \log^2 p/(M^2 \log^2 r))$ power residue symbol evaluations and $\mathcal{O}(M^2 \log r)$ memory. However, for large values of r, there exists a better attack which is detailed in the next section.

7.3 Attacks for Large r

We first describe a very simple attack on the linear r-th power residue Legendre PRF that requires $\mathcal{O}(p/r)$ power residue symbol evaluations. In the following, denote the subgroup of $(p-1)/r$-th roots of unity of $\mathbb{F}_p^\times$ by $\mathbb{G}$. That is,

$$\mathbb{G} = \{x \in \mathbb{F}_p^\times \mid x^{(p-1)/r} = 1\}.$$

Remark that $\mathbb{G}$ is generated by g^r, where g is any generator of $\mathbb{F}_p^\times$.

By querying $L_k^{(r)}(0)$, the attacker immediately learns $l^{(r)}(k)$, the power residue symbol of $k \in \mathbb{F}_p$. We observe that this single query already narrows down the set of possible values for k to at most $(p-1)/r$ elements of $\mathbb{F}_p$. Indeed, from Definition 4, k is contained in the coset $g^s\mathbb{G}$, where g is any generator of $\mathbb{F}_p^\times$ and s is equal to $l^{(r)}(k)$. Therefore, an attacker can just go through all of these elements and check each candidate. Since, on average, only $\mathcal{O}(1)$ power residue symbols must be computed to check the validity of a candidate key, the attack requires $\mathcal{O}(p/r)$ power residue symbols evaluations. The attack requires a generator g, which can be precomputed in probabilistic subexponential time by factoring $p-1$.

We now explain a more general attack that requires $\mathcal{O}(p \log^2 p/(Mr \log^2 r))$ power residue symbol evaluations and $\mathcal{O}(M \log r)$ memory. The attack is similar to the table-based collision search from Section 3.1. A speed-up of a factor r is obtained by querying the PRF at more carefully chosen arithmetic L-sequences. Let $m = \lceil \log p / \log r \rceil$ and $M < p/r$. The attack proceeds as follows:

1. For M/m distinct values $a \in \mathbb{G}$, store each pair $(L_k^{(r)}(a[m]), a)$ in a table $\mathcal{T}$. Furthermore, query the PRF to get the value $s = L_k^{(r)}(0)$.
2. Sample x uniformly at random from the coset $g^s\mathbb{G}$ until $(L_0^{(r)}(x+[m]), a) \in \mathcal{T}$ for some value a. For each entry $(L_0^{(r)}(x+[m]), a) \in \mathcal{T}$ corresponding to

such a collision, a candidate key is recovered as $\tilde{k} = xa$. By a variant of Assumption 1, the number of such candidate keys will be at most $\mathcal{O}(1)$.

The first step of the above attack uses $M = m \cdot (M/m)$ queries to $L_k^{(r)}$ and needs $\mathcal{O}(M \log r)$ memory to store the table $\mathcal{T}$. The key k is found when, in the second step, the attacker samples an x such that k/x is one of the a-values stored in the table. On average, $|\mathbb{G}|/(M/m) = \mathcal{O}(pm/(Mr))$ iterations of the second step are required in order to find a candidate key. Since each iteration requires m power residue symbol computations to evaluate $L_0^{(r)}(x + [m])$, it follows that the total time-complexity of the attack consists of $\mathcal{O}(M)$ storage operations and $\mathcal{O}(pm^2/(Mr)) = \mathcal{O}(p \log^2 p/(Mr \log^2 r))$ power residue symbol evaluations.

8 Implementation Results

This section discusses several aspects of our implementation of the attack from Section 3.3 that we applied to the key recovery puzzles proposed by the Ethereum foundation [Fei19b]. Using the attack from Section 3, we managed to solve three out of six challenges (including the test instance with a 40-bit prime). A summary of the instance parameters and the time and memory requirements of the attack is given in Table 2.

The source code of our implementation is publicly available at

`https://github.com/cryptolu/LegendrePRF`

Table 2. Parameters of the concrete challenges proposed by the Ethereum foundation [Fei19b]. For all instances, the first $M = 2^{20}$ consecutive PRF outputs were given. For the first three instances, the running time and peak memory usage is given, for the three hardest instances an estimation of time is provided (marked by †). All experiments were performed on a Dell C6420 server with two Intel Xeon Gold 6132 CPUs clocked at 2.6 GHz and 128 GB of RAM.

p	Security level[5] (bits)	Time (CPU-hours)	Memory / thread (GB)	Key
$2^{40} - 87$	20	¡ 0.001	¡ 1	`4e2dea1f3c`
$2^{64} - 59$	44	1.5	3	`90644c931a3fba5`
$2^{74} - 35$	54	1500	3	`384f17db02976dcf63d`
$2^{84} - 35$	64	2^{21}†	3	
$2^{100} - 15$	80	2^{37}†	3	
$2^{148} - 167$	128	2^{65}†	3	

We compiled our C++ implementation of the attack using Clang 6.0.0 and executed it on a Dell C6420 server with two Intel Xeon Gold 6132 CPUs clocked at 2.6 GHz (28 cores) and 128 GB of RAM. The optimizations described in Section 3.4 allow to significantly reduce the required memory and the number of evaluations of the Legendre symbol. As a result, the table lookups are the bottleneck in our implementation. On average, a single thread required $0.08\,\mu s$ to compute and check a single 64-bit sequence. As discussed below, we expect to compute $p/2^{28}$ sequences on average before the key is recovered. Hence, the required CPU time to solve a challenge with a prime p and 2^{20} bits of PRF output can be estimated as $p/2^{28} \times 0.08\,\mu s$. The required memory is 1 GB per server and an additional 3 GB per thread. The parameters can be modified to reduce the memory without significantly decreasing the performance.

For the first three instances we successfully recovered the secret key of the PRF in a timespan close to our estimation. The corresponding keys are given in Table 2. The third instance was solved in under two hours using a cluster of 40 nodes with the described configuration. Further details about the main steps of the attack are provided below.

Step 1: Processing the PRF Output As a first step we compute the set $\mathcal{T}$ consisting of all arithmetic sequences extracted from the sequence $L_k([2^{20}])$ given in the challenge. We chose to store sequences of length $m = 64$ since this length provides an acceptable rate of false-positives and enables to efficiently process sequences as 64-bit words. As a result, the set $\mathcal{T}$ contains approximately $M^2/(2m^2) = 2^{27}$ of such words-sequences.

A straightforward way to implement a set is by using a hash table, which has a constant amortized time-complexity for membership testing. However, this constant time may be quite large in practice, especially in the case of large tables. Random memory accesses are often the main bottleneck. In our case, the set $\mathcal{T}$ is never modified after its creation. To exploit this fact, we sort the elements of $\mathcal{T}$ and we store them in an array. Then, we compute membership queries in batches. First, we collect a large amount of membership queries and we sort them. Then, we scan through the two sorted arrays checking for collisions. The bottleneck in this approach is represented by the sorting step of each batch of membership queries. The described set $\mathcal{T}$ contains 2^{27} 64-bit words and the corresponding sorted array requires 1 GB of memory. An extra 1 GB of memory is used to store information required for the key recovery. Note that the set $\mathcal{T}$ and the extra information are shared among all threads that are used to parallelize the workload of the next step.

Step 2: Random Sampling The second and main step of the attack consists of sampling sequences $L_0(c + [m])$ for randomly chosen c and checking if they

[5] Expected security level (conservative estimate) prior to this work.

collide with an entry of $\mathcal{T}$. Note that the reversed sequence $L_0(c+[m])$ is checked if it is lexicographically smaller.

For a uniformly chosen $c \in \mathbb{F}_p$ we compute a long sequence $L_0(c + [t])$ and we extract a large amount of m-bit sequences from it. More precisely, for all $b \in (1 + [2^8])$ and $a \in [t - b(m - 1)]$, we extract $L_0(c + a + b[m])$. The upper-bound for b is chosen as 2^8 since it is enough to make the time spent on computing Legendre symbols negligible. Furthermore, all these sequences can be computed on the fly by storing only the last sequence per pair (b, a). Indeed, for a large enough $i \in \mathbb{Z}$, after expanding the computed sequence $L_0(c + [i - 1])$ by one Legendre symbol $L_0(c+i)$ we obtain a new sequence $L_0(c+i-b(m-1)+b[m])$ for each b. In other words, we obtain 2^8 sequences from each single consequent Legendre symbol computation.

As described above, the computed sequences are accumulated and checked in batches for a collision with the set $\mathcal{T}$. Each batch is sorted using base-2^8 radix sort and collisions are checked using a linear scan through the sorted batch and the sorted array of $\mathcal{T}$. In the case of a collision, a key candidate is recovered and checked against extra bits from the given PRF output.

Note that this step can be efficiently parallelized. Each thread starts with a uniformly random $a \in \mathbb{F}_p$ and proceeds as described above. After a predetermined amount of steps, a new value for a can be chosen to ensure a sufficiently uniform coverage of the possible offsets of the sequences.

9 Conclusions

In Section 3, a new attack on the Legendre PRF was presented. It is of particular interest in the low-data setting. Specifically, given $M \leq \sqrt[4]{p}$ queries, our attack recovers the key using $\mathcal{O}(p \log^2 p/M^2)$ Legendre symbol evaluations. The practical relevance of this result was demonstrated by solving the first two Legendre PRF challenges set out by the Ethereum foundation [Fei19b]. Several aspects of our implementation of the attack were discussed in Section 8.

In Section 4, it was shown how the technique from Section 3 yields improved attacks on the higher-degree generalization of the Legendre PRF. Further attacks on the higher-degree case were given in Section 5, where a large class of weak keys was revealed. Keys from this class can be recovered using $\mathcal{O}(p^{\lfloor d/2 \rfloor} d \log p)$ Legendre symbol evaluations and $\mathcal{O}(p^{\lceil d/2 \rceil} d \log p)$ memory. Further improvements to the memory usage, based on a reduction to the unique k-XOR problem, were also discussed. These weak key attacks can be prevented by choosing the key such that the corresponding monic polynomial is irreducible.

In addition to the above, we provided the first security analysis of the Jacobi and power-residue generalizations of the Legendre PRF. These extensions were first suggested – for the Legendre pseudorandom generator – at CRYPTO 1988 by Damgård [Dam90]. It was demonstrated in Section 6 that the key of a Jacobi

PRF can be recovered with time-complexity proportional to the time-complexity of key-recovery on the Legendre PRF for each of the prime factors of the modulus separately. This result eliminates the potential efficiency benefits offered by Jacobi symbols.

Power residue symbols were considered in Section 7. The low-data attack from Section 3 equally applies in this setting, but we provide an additional attack that preforms better for large power residue symbols. Specifically, for r-th power residue symbols and given $M \leq \sqrt{p}$ queries, our key-recovery attack requires $\mathcal{O}(p \log^2 p/(rM \log^2 r))$ power residue evaluations and $\mathcal{O}(M)$ memory.

References

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.

Ala96. N. Aladov. Sur la distribution des résidus quadratiques et non-quadratiques d'un nombre premier p dans la suite $1, 2, \ldots, p-1$. *Matematicheskii Sbornik*, 18(1):61–75, 1896.

BJLM13. Daniel J Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. Quantum algorithms for the subset-sum problem. In *International Workshop on Post-Quantum Cryptography*, pages 16–33. Springer, 2013.

BZ10. Richard P Brent and Paul Zimmermann. An $\mathcal{O}(M(n) \log n)$ algorithm for the Jacobi symbol. In *International Algorithmic Number Theory Symposium*, pages 83–95. Springer, 2010.

CJM02. Philippe Chose, Antoine Joux, and Michel Mitton. Fast correlation attacks: An algorithmic point of view. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 209–221. Springer, Heidelberg, April / May 2002.

Dam90. Ivan Damgård. On the randomness of legendre and jacobi sequences. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 163–172. Springer, Heidelberg, August 1990.

Dav31. Harold Davenport. On the distribution of quadratic residues (mod p). *Journal of the London Mathematical Society*, 1(1):49–54, 1931.

Dav39. Harold Davenport. On character sums in finite fields. *Acta Mathematica*, 71(1):99–121, 1939.

Fei19a. Dankrad Feist. Cryptanalyzing the Legendre PRF. CRYPTO rump session talk, August 2019.

Fei19b. Dankrad Feist. Legendre pseudo-random function. `https://legendreprf.org`, 2019. Accessed: 2019-11-18.

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 2016*, pages 430–443. ACM Press, October 2016.

Jac06. Ernst Erich Jacobsthal. *Anwendungen einer Formel aus der Theorie der quadratischen Reste*. PhD thesis, Friedrich-Wilhelms Universität zu Berlin, 1906.

Kho19. Dmitry Khovratovich. Key recovery attacks on the Legendre PRFs within the birthday bound. Cryptology ePrint Archive, Report 2019/862, 2019. `https://eprint.iacr.org/2019/862`.

MOM92. Hikaru Morita, Kazuo Ohta, and Shoji Miyaguchi. A switching closure test to analyze cryptosystems. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 183–193. Springer, Heidelberg, August 1992.

NPS19. María Naya-Plasencia and André Schrottenloher. Optimal merging in quantum k-xor and k-sum algorithms. Cryptology ePrint Archive, Report 2019/501, 2019. `https://eprint.iacr.org/2019/501`.

Tao15. Terence Tao. Cycles of a random permutation and irreducible factors of a random polynomial. `https://terrytao.wordpress.com/2015/07/15/cycles-of-a-random-permutation-and-irreducible-factors-of-a-random-polynomial/`, 2015. Accessed: 2019-11-18.

VBCG14. S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos. Management of an academic HPC cluster: The UL experience. In *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*, pages 959–967, Bologna, Italy, July 2014. IEEE.

vDH00. Wim van Dam and Sean Hallgren. Efficient quantum algorithms for shifted quadratic character problems. *arXiv preprint quant-ph/0011067*, 2000.

vOW94. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, and Ravi S. Sandhu, editors, *ACM CCS 94*, pages 210–218. ACM Press, November 1994.

vS98. R. von Sterneck. Sur la distribution des résidus et des non-résidus quadratiques d'un nombre premier. *Matematicheskii Sbornik*, 20(2):269–284, 1898.

Wag02. David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Heidelberg, August 2002.

Wei48. André Weil. On some exponential sums. *Proceedings of the National Academy of Sciences of the United States of America*, 34(5):204, 1948.