Threshold and Proactive Pseudo-Random Permutations*

Yevgeniy Dodis[1]**, Aleksandr Yampolskiy[2]***, and Moti Yung[3]

[1] New York University, Department of Computer Science,
251 Mercer Street, New York, NY 10012, USA,
dodis@cs.nyu.edu
[2] Yale University, Department of Computer Science,
51 Prospect Street, New Haven, CT 06520, USA,
aleksandr.yampolskiy@yale.edu
[3] RSA Laboratories and Columbia University, Department of Computer Science,
1214 Amsterdam Avenue, New York, NY 10027, USA,
moti@cs.columbia.edu

Abstract. We construct a reasonably efficient threshold and proactive pseudo-random permutation (PRP). Our protocol needs only $O(1)$ communication rounds. It tolerates up to $(n-1)/2$ of n dishonest servers in the semi-honest environment. Many protocols that use PRPs (*e.g.*, a CBC block cipher mode) can now be translated into the distributed setting. Our main technique for constructing invertible threshold PRPs is a distributed Luby-Rackoff construction where both the secret keys *and* the input are shared among the servers. We also present protocols for obliviously computing pseudo-random functions by Naor-Reingold [37] and Dodis-Yampolskiy [24] with shared input and keys.

Key words. Distributed Block Ciphers, Distributed Luby-Rackoff Construction, Oblivious Pseudo-Random Functions, Threshold Cryptography.

1 Introduction

Block ciphers are familiar cryptographic tools, which transform blocks of plaintext into blocks of ciphertext of the same length. The DES (U.S. Data Encryption Standard) is a well-known example of a block cipher, which was, until recently, used by many financial firms to protect online transactions. Traditionally, **pseudo-random permutations** (PRPs) have been used to model secure block ciphers [33]. They map l-bit inputs into unique l-bit outputs that appear random to parties who lack the secret key. A close relative of the PRP is a

* An extended abstract of this paper is to appear in the proceedings of the Third Theory of Cryptography Conference (TCC 2006).

** Supported in part by NSF career award CCR-0133806 and NSF grant CCR-0311095.

*** Supported by NSF grants CCR-0098078,ANI-0207399,CNS-0305258, and CNS-0435201.

pseudo-random function (PRF), which needs not be invertible, but whose outputs also look like random bit-strings without the secret key [27].

Motivation. The security of these functions relies on the owner of the secret key, who has a primary responsibility of keeping the key safe. Alas, it is not always realistic to put all trust into a single party. The area of **threshold cryptography** deals exclusively with sharing the ability to perform cryptographic operations between a set of n servers [20]. A long line of research produced many distributed protocols that are more efficient than generic multi-party solutions when used for public key encryption [17,40,41], digital signatures [19,21,22,26], key generation [1,8,25], pseudo-random functions [10,36,38], and other applications. The extra security and increased availability of constructions justify the added complexity. The pseudo-random permutation is the only primitive that is still missing from this long list.

Several initial attempts [9,34] gave a very basic sharing structure with many limitations and drawbacks.[4] The question of constructing a threshold PRP was yet left open. In this paper, we resolve this problem. Many protocols are defined for PRPs (block ciphers) and, when needed, can now be readily translated into the distributed setting. This makes sense for sensitive operations like key-encrypting-key in the Key Distribution Center [36]. Applications such as distributed remotely keyed authenticated encryption and CBC encryption mode become possible, since they require a PRP as a building block (regular PRFs do not suffice).

We focus on implementing the Luby-Rackoff construction [33] as a method for building PRPs. It uses the **Feistel permutation** for function F (denoted $\bar{F}$), which sends a $2l$-bit string (x^L, x^R) to a $2l$-bit string $(x^R, x^L \oplus F(x^R))$. Luby and Rackoff showed that a composition of four Feistel permutations (denoted $\Psi(F_1, F_2, F_3, F_4) = \bar{F}_1 \circ \cdots \circ \bar{F}_4$) is a secure $2l$-bit PRP when F_i are independent l-bit PRFs. While a sequential composition of PRFs to build a sequential PRP is generic, there is a major technical difficulty in the distributed Luby-Rackoff construction. Particularly, the difficult part is that if one uses a PRF as an intermediate round function, then not just the secret key, but also the output needs to be kept distributed to assure the security of the entire Luby-Rackoff construction. At the same time, the computation needs to continue and compute on these shares, which means that we need to compute on shared inputs as well.

Our Results. This paper describes an $O(1)$ round distributed protocol for evaluating $\Psi(F_1, F_2, F_3, F_4)$, which results in a **threshold pseudo-random permutation**. Our protocol invokes the multiplication protocol for the underlying secret sharing scheme $O(mn + m \log m)$ times, where n is the number of servers and m is the maximum input length. It tolerates up to $\tau = \lfloor (n-1)/2 \rfloor$ dishonest

[4] They showed how to build rather inefficient cascade ciphers $E_k(x) = g_{k_m}(\dots g_{k_2}(g_{k_1}(x)))$, where $g(\cdot)$ is itself a secure cipher, by sharing a sequence of keys in a special way. For τ-out-of-n sharing, the number of keys and composition layers is on the order of $\binom{n}{\tau}$, which is exponential for most $\tau = \omega(1)$.

servers in the semi-honest model, which is consistent with some prior work on distributed PRFs [36] and multiparty tools [1, 12, 15] used in our constructions. It can be made robust using standard techniques [42] and, as we show, can be amended to ensure **proactive security** [31].

As we have explained, intermediate Feistel values arising after each round of the Luby-Rackoff construction must be kept secret, yet we must evaluate the PRFs F_i on them. Unfortunately, prior distributed PRF constructions [10,36,38] are inapplicable to our problem, because they require the PRF input to be publicly known. We give two protocols for distributed computation of PRFs by Naor-Reingold [37] and Dodis-Yampolskiy [24] when both the secret keys *and* the input are shared among the servers. Effectively, we implement **oblivious distributed PRFs**, where servers do not learn what the input is, yet blindly compute the PRF value.

We note that, theoretically, we can always use general multi-party techniques [5] to distribute the computation of a PRP. Until recently, this was not a viable option. These techniques either (i) required a linear number of communication rounds (in the circuit depth) [5, 44] or (ii) ran in $O(1)$ rounds but used expensive zero-knowledge proofs for each gate of the circuit [4]. A recent improvement by Damgård *et al.* [16] allows to securely evaluate any circuit C in $O(1)$ rounds using $O(|C|n)$ cryptographic operations ($|C|$ is the circuit size). If we distribute the DES circuit (which is believed to be a PRP) using Damgård *et al.*'s techniques, we obtain comparable efficiency to our threshold PRP.[5] Our protocol is thus fairly practical. In addition, it has theoretical value in and of itself and could be of independent interest in other fields.

Overview of Our Construction. In our protocol, servers hold Shamir shares [43] of secret keys SK_i to PRFs F_i used in the Luby-Rackoff (LR) construction of a PRP $\Psi(F_1, F_2, F_3, F_4)$. The untrusted user who wants to compute the PRP's value broadcasts his input x to the servers. Servers somehow verify that the user is entitled to evaluate the PRP and engage in an interactive protocol, which terminates with shares of the PRP's value.

Our round functions F_i are based on a PRF by Dodis-Yampolskiy [24]. We chose this PRF because it possesses useful algebraic properties and can be computed in $O(1)$ rounds. Given an l-bit input $x = x_1x_2 \ldots x_l$ (which can be viewed as an element of $\mathbb{Z}_Q$) and a secret key $SK \in \mathbb{Z}_Q$, the PRF value is $F_{SK}(x) = g^{1/(x+SK)}$. Here, g is the generator of a group in which the **decisional Diffie-Hellman inversion** (y-DDHI) problem is hard. The y-DDHI problem asks: "given $(g, g^x, \ldots, g^{(x^y)}, R)$ as input, to decide whether $R = g^{1/x}$ or not." It appears hard in a quadratic residues subgroup G_Q of $\mathbb{Z}_P^*$ ($P = 2Q + 1$) for sufficiently large primes P, Q.

[5] In fact, for realistic settings, our algorithm performs better. The full DES circuit contains about $|C| \approx 16000$ Boolean gates [6]. Let the group size be a $m = 1024$ bit prime and the number of servers be $n = 100$. Our protocol performs roughly $(mn + m \log m) \cdot (m^2 n + mn^2 \log n) \approx 1.95 \times 10^{13}$ bit operations, while Damgård *et al.*'s protocol [16] performs $(|C|n) \cdot (mn^2 \log n) \approx 10.9 \times 10^{13}$ operations to compute the DES circuit.

Dodis and Yampolskiy showed that $F_{SK}(\cdot)$ is secure only for inputs of small length $l = \omega(\log k)$, which makes it unsuitable for the LR construction, whose round functions must accept longer $l = \Theta(k)$ bit inputs (k is the security parameter). In this paper, we assume subexponential hardness of the y-DDHI problem. This immediately allows us to support inputs of size $a = \Theta(k^\delta)$ for some small $\delta \approx 1/3$. We can shrink the input to the LR construction from $l = \Theta(k)$ bits down to $a = \Theta(k^\delta)$ bits using an ϵ-universal hash function $h_i(x) = (ix \bmod Q) \bmod 2^a$. We thus get a new PRF $F'_{i,SK}(x) = F_{SK}(h_i(x))$, which can be used in the (centralized) LR construction.

We distribute the LR construction using well-known multiparty tools of addition, multiplication, inversion, etc. [1,3,5]. We rely heavily on an $O(1)$ round protocol by Damgård *et al.* [15], which computes shares of bits of $x \in \mathbb{Z}_P$ from shares of x. This protocol allows us to efficiently perform modular reduction, exponentiation, and truncation of shared values.

We can compute the PRF $F'_{i,SK}(x)$ with shared input x and keys (i, SK) as follows. Computing the ϵ-universal hash $h_i(x) = (ix \bmod Q) \bmod 2^a$ amounts to a single multiparty multiplication, followed by a call to Damgård *et al.*'s protocol to extract the trailing a bits. We can also distribute the computation of $F_{SK}(x) = g^{1/(x+SK)}$ because it is well-known how to do multiparty addition, inversion, and exponentiation. As a result, we obtain a sharing of $F'_{i,SK}(x)$, a random group element in G_Q, whereas we need a sharing of a random l-bit string. We can use a deterministic extractor $E(x) = (x^{(P+1)/4} \bmod P) \bmod 2^l$ to convert this group element into a random l-bit string. Computing this extractor distributively entails a single distributed exponentiation followed by a call to Damgard *et al.*'s protocol to extract l bits.

Armed with a protocol for computing the PRF $F'_{i,SK}(\cdot)$, we can distribute a single Feistel permutation, which maps (x^L, x^R) into $(x^R, x^L \oplus F'_{i,SK}(x^R))$. The only missing link is how to XOR shares of PRF's bits with shares of input's bits. Given shares of bits $b_1, b_2 \in \{0,1\}$, we can get a share of $b_1 \oplus b_2$ by distributively computing $(b_1 + b_2) \cdot (2 - (b_1 + b_2))$. This completes our calculation. We obtain a threshold PRP by iterating the distributed Feistel permutation four times, cross-feeding its outputs to inputs.

Paper Organization. The remainder of the paper is organized as follows. Section 2 reviews some preliminaries and defines our system model. In Section 3, we give distributed protocols for evaluating pseudo-random functions by Naor-Reingold [37] and Dodis-Yampolskiy [24] when both the keys and the input are shared. Then in Section 4, we present our threshold PRP construction. In Section 5, we describe a distributed exponentiation protocol, which is used throughout the paper. Some practical applications of our threshold PRP appear in Section 6. We conclude in Section 7.

2 Preliminaries

In this section, we discuss some basic definitions and assumptions.

2.1 Our Model

Let k be a security parameter. We consider n computationally bounded servers $P_1, \ldots, P_n$, which are connected by secure and authentic channels[6]. Our protocols are secure against a static, honest-but-curious adversary who controls up to $\tau = \lfloor (n-1)/2 \rfloor$ servers. This threshold results from the multiplication protocol by Ben-Or *et al.* [5], which is used throughout the paper. We prove security in the framework by Canetti [11]. In the honest-but-curious setting, privacy is preserved under non-concurrent modular composition of protocols. This composition theorem will be the main source of our privacy proofs.

2.2 Notation

The notation in this paper is adapted from [1, 15]. We define $\mathbb{Z}_P$ as the set $\{0, \ldots, P-1\}$. We denote additive shares over $\mathbb{Z}_P$ of a value $a \in \mathbb{Z}_P$ by $\langle a \rangle_1^P, \ldots, \langle a \rangle_n^P \in \mathbb{Z}_P$; *i.e.*, $a = \sum_{j=1}^{n} \langle a \rangle_j^P \bmod P$. Meanwhile, we denote Shamir shares [43] of $a \in \mathbb{Z}_P$ by $[a]_1^P, \ldots, [a]_n^P \in \mathbb{Z}_P$; *i.e.*, $a = \sum_{j=1}^{\tau} \lambda_j [a]_j^P \bmod P$, where τ is the threshold and λ_j are the Lagrange coefficients.

We denote protocols as follows: the term $[a]_j^P \leftarrow \texttt{PROTOCOL}([b]_j^P, c)$ means that server P_j executes the protocol `PROTOCOL` with local input $[b]_j^P$ and public input c. As a result of the protocol, it gets back local output $[a]_j^P$. In all cases, the local inputs and outputs will be Shamir shares over the appropriate field.

2.3 Building Blocks

We review some standard tools for multiparty computation that are used throughout the paper. All these protocols require $O(1)$ rounds of communication. We measure their running time in terms of bit operations in $m = \lceil \log_2 P \rceil$ (the modulus length) and n (the number of servers). Below, we use $\mathcal{B}$ as a shorthand for $O(nm^2 + mn^2 \log n)$.

Sharing a secret. To compute a Shamir sharing of $x \in \mathbb{Z}_P$ over $\mathbb{Z}_P$, player P_j chooses random coefficients $\alpha_k \in \mathbb{Z}_P$ for $k = 1, \ldots, \tau$. He then sends $[x]_i^P = x + \sum_{k=1}^{\tau} \alpha_k \cdot i^k \bmod P$ to player P_i. We denote this protocol by $\texttt{RANDSS}(x, \mathbb{Z}_P)$; it takes $O(n^2 m \log n)$ bit operations.

Basic operations. Addition and multiplication of a constant and a Shamir share can be done locally. Hence, $[x]_j^P + c \bmod P$ is a polynomial share of $x + c \bmod P$ and $c \cdot [x]_j^P \bmod P$ is a share of $xc \bmod P$. These operations take $O(m)$ and $O(m^2)$ bit operations, respectively. Similarly, we can compute $[x]_j^P + [y]_j^P \bmod P$, which is a share of $x + y \bmod P$. Addition requires $O(m)$ bit operations.

[6] Such channels can be implemented using public-key encryption and digital signatures.

Multiplication. We note that a product of polynomially many shared secrets $x_1, \ldots, x_s \in \mathbb{Z}_P^*$ can be computed in constant rounds [3, 15]. We denote this protocol by $\mathtt{MUL}([x_1]_j^P, \ldots, [x_s]_j^P)$; it uses $O(s\mathcal{B})$ bit operations.

Conversion between bit shares. Given Shamir shares of a single bit $b \in \{0, 1\}$ in $\mathbb{Z}_P$, we may need to obtain its shares in $\mathbb{Z}_Q$. We can do this as follows. First, each server P_j locally computes $[b']_j^P \leftarrow -2 \cdot [b]_j^P + 1 \pmod{P}$ to convert the bit from a $0/1$ to a $1/-1$ encoding. Next, P_j chooses a random $b_j \in \{1, -1\}$ and shares it among servers in both $\mathbb{Z}_P$ and $\mathbb{Z}_Q$. He computes $[b'']_j^P \leftarrow \mathtt{MUL}([b']_j^P, [b_1]_j^P, \ldots, [b_n]_j^P)$ and reveals it for all servers to reconstruct b''. Finally, P_j multiplies $b'' \pmod{Q}$ by its share of $\mathtt{MUL}([b_1]_j^Q, \ldots, [b_n]_j^Q)$ and converts the result to a $0/1$ encoding. The protocol requires $O(1)$ rounds and $O(n\mathcal{B})$ bit operations.

Bit representation. Let $x \in \mathbb{Z}_P$ be a shared secret (written $x_m \ldots x_1$ in binary). In some situations, we will need to obtain Shamir shares of the bits of x. For this, we will use a protocol by Damgård *et al.* [15], denoted $([x_1]_j^P, \ldots, [x_m]_j^P) \leftarrow \mathtt{BITS}([x]_j^P)$, which uses $O((m \log m)\mathcal{B})$ bit operations.

Occasionally, we will need to compute shares of a least significant bits of $x \in \mathbb{Z}_P$ in $\mathbb{Z}_Q$ (rather than in $\mathbb{Z}_P$). We will first run the $\mathtt{BITS}([x]_j^P)$ protocol and then convert each bit share from $\mathbb{Z}_P$ to $\mathbb{Z}_Q$. We denote this protocol by $([x_1]_j^Q, \ldots, [x_a]_j^Q) \leftarrow \mathtt{BITS}([x]_j^P, a, \mathbb{Z}_Q)$. It requires $O(1)$ rounds and $O((an + m \log m)\mathcal{B})$ bit operations.

Given bit-by-bit shares of $x \in \mathbb{Z}_P$, denoted $[x_1]_j^P, \ldots, [x_m]_j^P$, we can easily obtain shares of x by locally computing $[x]_j^P \leftarrow \sum_{i=1}^{m} 2^{i-1} \cdot [x_i]_j^P \bmod P$. This takes $O(m^3)$ bit operations.

Inversion. Let $x \in \mathbb{Z}_P$ be a shared secret. A protocol due to Bar-Ilan and Beaver [3], denoted by $\mathtt{INV}([x]_j^P)$, allows us to compute the shares of $x^{-1} \bmod P$. It takes an expected number of $O(\mathcal{B})$ bit operations.

Generating a random number. Occasionally, servers may need to jointly generate shares of a random number. A simple protocol, denoted $\mathtt{JRP}(\mathbb{Z}_P)$, accomplishes this in $O(mn^2 \log n)$ bit operations [1]. There also exists a protocol $\mathtt{JRPZ}(\mathbb{Z}_P)$ to jointly compute a sharing of zero modulo P in $O(mn^2 \log n)$ bit operations.

Exponentiation Some of our protocols require computing the shares of $x^y \bmod P$ when: (i) the exponent $y \in \mathbb{Z}_Q$ is shared, but the base is fixed; (ii) the base $x \in \mathbb{Z}_P$ is shared and the exponent is fixed; or (iii) both the base and the exponent are shared. We denote protocols for the above scenarios $\mathtt{EXP}_1(x, [y]_j^Q)$, $\mathtt{EXP}_2([x]_j^P, y)$, and $\mathtt{EXP}([x]_j^P, [y]_j^Q)$. These protocols run in $O(1)$ rounds and require, respectively, $O((mn + m \log m)\mathcal{B})$, $O(m^3 + n\mathcal{B})$, and $O(m^4 + (mn + m \log m)\mathcal{B})$ bit operations per player. We describe them later in Section 5.

3 Distributed Pseudo-Random Functions

In this section, we describe two distributed PRF constructions, where both the secret key and the input are shared. This will ensure that unscrupulous servers do not learn the results of intermediate Luby-Rackoff computations. In Section 3.1, we show how to do this for the PRF by Naor and Reingold [37]. Then in Section 3.2, we describe how to do this for the recently introduced PRF by Dodis and Yampolskiy [24].

Let the input size $l : \mathbb{N} \mapsto \mathbb{N}$ be a function computable in poly(k) time. Sometimes, for simplicity, we will write l for $l(k)$. The initial input for all servers is a triple (P, Q, g), where P, Q are large primes such that $P = 2Q + 1$ and $P \equiv 3 \bmod 4$. Here g is a generator of quadratic residues subgroup G_Q of $\mathbb{Z}_P^*$. The group $\mathbb{Z}_P^*$ must be sufficiently large, *i.e.*, $P \gg 2^k$. Such a triple can be publicly chosen without a trusted party by executing Bach's algorithm [2].

Both centralized PRFs take as input an l-bit message x, the secret key SK and output a random group element in G_Q. In our distributed PRFs, each server P_j receives a share of the secret key SK and l shares of bits of x.

3.1 Naor-Reingold PRF

The secret key $SK = (a_0, a_1, \ldots, a_l)$ consists of $l + 1$ random exponents in $\mathbb{Z}_Q$. Given an l-bit input $x = x_1 \ldots x_l$, the PRF $F_{SK}^{NR} : \{0,1\}^l \mapsto G_Q$ is defined as

$$F_{SK}^{NR}(x) = (g^{a_0})^{\prod_{i\,:\,x_i=1} a_i} .$$

This PRF was shown to be secure for polynomially sized inputs, $l(k) = \text{poly}(k)$, under the decisional Diffie-Hellman (DDH) assumption: "given (g, g^x, g^y) and $R \in G_Q$, it is hard to determine if $R = g^{xy}$ or not."

We can compute the PRF value recursively. Set $h_0 = g^{a_0}$. Then, for all $i = 1, \ldots, l$,

$$h_i = \begin{cases} h_{i-1}^{a_i} & \text{if } x_i = 1, \\ h_{i-1} & \text{otherwise.} \end{cases} \tag{1}$$

It is easily seen that the PRF value must be equal to h_l. This form is convenient for distributed computation when both the input x and the secret exponents a_i are shared. One problem here is that we need to implement an if-condition on secret input x. We can use a simple trick and rewrite Equation (1) as

$$h_i = h_{i-1}(1 - x_i) + h_{i-1}^{a_i} x_i \text{ for } x_i \in \{0, 1\}. \tag{2}$$

Computing the PRF value distributively amounts to several rounds of distributed multiplication and exponentiation:

Algorithm 1: A protocol $\texttt{PRF-NR}(([a_0]_j^Q, \ldots, [a_l]_j^Q), ([x_1]_j^P, \ldots, [x_l]_j^P))$ for distributed computation of $F_{SK}^{NR}(x)$.

1 $[0]_j^P \leftarrow \texttt{JRPZ}(\mathbb{Z}_P)$ $\triangleright$Servers jointly generate a sharing of 0 mod P
2 $[h_0]_j^P \leftarrow [0]_j^P + g \bmod P$ $\triangleright$And compute a share of generator g.
3 **for** $i \leftarrow 1$ **to** l $\triangleright$For all input bits i,
4 **do**
5 $\quad [r]_j^P \leftarrow \texttt{MUL}([h_{i-1}]_j^P, 1 - [x_i]_j^P)$
6 $\quad [s]_j^P \leftarrow \texttt{EXP}([h_{i-1}]_j^P, [a_i]_j^Q)$
7 $\quad [t]_j^P \leftarrow \texttt{MUL}([s]_j^P, [x_i]_j^P)$ $\triangleright$we compute shares of Equation (2)
8 $\quad [h_i]_j^P \leftarrow [r]_j^P + [t]_j^P$
9 **end**
10 **return** $[h_l]_j^P$ $\triangleright$Return a share of the PRF value.

Proving security of this protocol is straightforward given the security of its sub-protocols by the composition theorem. The size of the secret key is proportional to the length of the input. What is worse, this protocol requires $O(l)$ rounds of communication. The running time is dominated by l calls to exponentiation protocol in line 6, yielding $O(m^4 l + (mn + m \log m)\mathcal{B}l)$. bit operations per player.

3.2 Dodis-Yampolskiy PRF

The pseudo-random function $F_{SK}^{DY} : \{0,1\}^{l(k)} \mapsto G_Q$ $(|Q| > l)$ is as follows. Given an l-bit input x (which can also be thought of as an element in $\mathbb{Z}_Q$) and the secret key $SK \in \mathbb{Z}_Q$, the function value is $F_{SK}^{DY}(x) = g^{1/(x+SK)}$ [24]. Dodis-Yampolskiy's proof of security relied on an unorthodox **q-decisional Diffie-Hellman inversion** (q-DDHI) assumption: "given the tuple $\left(g, g^x, \ldots, g^{(x^q)}\right)$ and $R \in G_Q$ as input, it is hard to decide whether $R = g^{1/x}$ or not." Specifically, they showed:

Theorem 1 (Dodis-Yampolskiy). *Suppose an attacker who runs for $s(k)$ steps cannot break the $2^{l(k)}$-DDHI assumption in group G_Q with advantage $\epsilon(k)$. Then no algorithm running in less than $s'(k) = s(k)/(2^{l(k)} \cdot poly(k))$ steps can distinguish $F_{SK}^{DY}(\cdot)$ from a random function with advantage $\epsilon'(k) = \epsilon(k) \cdot 2^{l(k)}$.*

Because the security reduction is rather loose, we can construct PRFs only with small superlogarithmic input $l(k) = \omega(\log k)$. Unfortunately, "as is" this PRF is unsuitable for use in the Feistel transformation. A Feistel transformation uses length-preserving PRFs which map $l(k) = \text{poly}(k)$ input bits to $l(k)$ pseudo-random bits. In theory, small inputs are not a problem. We can either (1) shrink the inputs using a collision-resistant hash function [14] or (2) utilize the generic tree construction [27] to extend the input range. However, when we need to distribute the computation of this PRF between different servers, neither of these options becomes acceptable. As of today, we do not know how to efficiently distribute collision-resistant hash functions. And if we decide to utilize the generic tree construction, then we might as well use the Naor-Reingold PRF from the start.

Instead, we assume subexponential hardness of the q-DDHI assumption in G_Q; that is, we suppose that there is no way to break the q-DDHI assumption except by computing the discrete logarithm of g^x in $\mathbb{Z}_P^*$. The fastest algorithm for computing discrete logarithms modulo P runs in time roughly $\exp((1+o(1))\cdot\sqrt{\log P}\sqrt{\log\log P})$ [13]. It seems reasonable to assume that no algorithm running in time less than $s(k) = 2^{k^{\epsilon_2}}$ (for some small $\epsilon_2 \approx \frac{1}{3}$) can break the q-DDHI assumption. Formally:

Definition 1 (strong DDHI assumption). *We say that the strong DDHI assumption holds in G_Q if there exist $0 < \epsilon_1 < \epsilon_2$ such that for all probabilistic families of Turing machines $\{A_k\}_{k\in\mathbb{N}}$ with running time $O(2^{k^{\epsilon_2}})$ and $q \leq 2^{k^{\epsilon_1}}$, we have:*

$$\Big| \Pr_x\left[\mathcal{A}_k(g, g^x, \ldots, g^{(x^q)}, R) = 1 \mid R \leftarrow g^{1/x}\right] - \Pr_x\left[\mathcal{A}_k(g, g^x, \ldots, g^{(x^q)}, R) = 1 \mid R \xleftarrow{\$} G_Q\right] \Big| \leq poly(k)/2^{k^{\epsilon_2}},$$

where the probability is taken over the coin tosses of $\mathcal{A}_k$ and the random choice of $x \in \mathbb{Z}_Q^*$ and $R \in G_Q$.

By Theorem 1, the strong DDHI assumption immediately allows us to support inputs of size k^{ϵ_1} for small $\epsilon_1 > 0$.

What we need is a shrinking hash function, which maps long $l(k) = k$ bit inputs to smaller $a(k) = k^{\epsilon_1}$ bit inputs, which can be used as an input to $F_{SK}^{DY}(\cdot)$. A typical tool used for this purpose is a family of δ-universal hash functions $\mathcal{H} = \{h_i : \{0,1\}^l \mapsto \{0,1\}^a\}_{i\in\mathbb{Z}_Q^*}$.[7] The simplest such construct is

$$h_i(x) = (ix \bmod Q) \bmod 2^a,$$

where the collision probability $\delta = 1/2^a = 1/2^{k^{\epsilon_1}}$ is the best we can hope for.

We can thus define a new function $F' : \{0,1\}^l \mapsto G_Q$ as

$$F'_{SK,i}(x) = F_{SK}^{DY}(h_i(x)),$$

which is easily seen to be a secure PRF for polynomially sized inputs using a standard hybrid argument.

This new PRF can be used in the Feistel transformation. We now describe how to distribute its computation:

[7] We say that a hash family is δ-universal if, for all distinct inputs $x, x' \in \{0,1\}^l$, we have $\Pr_i[h_i(x) = h_i(x')] \leq \delta$.

Algorithm 2: A protocol $\texttt{PRF-DY}([i]_j^Q, [SK]_j^Q, [x_1]_j^Q, \ldots, [x_l]_j^Q)$

1 $[x]_j^Q \leftarrow \sum_{i=0}^{l-1} 2^i \cdot [x_{i+1}]_j^Q \bmod Q$ ▷Encode input x as an element in $\mathbb{Z}_Q^*$.
2 $[r]_j^Q \leftarrow \texttt{MUL}([i]_j^Q, [x]_j^Q)$ ▷Then hash it to $ix \bmod Q$.
3 $a \leftarrow \lfloor l^{1/3} \rfloor$ ▷Shrinking factor $a = l^{1/3}$.
4 $([r_1]_j^Q, \ldots, [r_a]_j^Q) \leftarrow \texttt{BITS}([r]_j^Q, a, \mathbb{Z}_Q)$ ▷Chop all but a least significant bits.
5 $[\tilde{x}]_j^Q \leftarrow \sum_{i=0}^{a-1} 2^i \cdot [r_{i+1}]_j^Q \bmod Q$
6 $[s]_j^Q \leftarrow [\tilde{x}]_j^Q + [SK]_j^Q \bmod Q$ ▷A share of $(\tilde{x} + SK)$.
7 $[t]_j^Q \leftarrow \texttt{INV}([s]_j^Q)$ ▷Invert the share into $1/(\tilde{x} + SK)$.
8 $[y]_j^P \leftarrow \texttt{EXP}_1(g, [t]_j^Q)$ ▷Exponentiate to get shares of $g^{1/(\tilde{x}+SK)}$.
9 **return** $[y]_j^p$

The security of the protocol again follows by composition theorem from security of its subcomponents. Unlike Algorithm 1, this algorithm uses $O(1)$ rounds of communication. However, it relies on a rather strong complexity assumption. Line 8 dominates the running time. It requires $O((mn + m \log m)\mathcal{B})$ bit operations, which is more than l times cheaper than the Naor-Reingold distributed protocol.

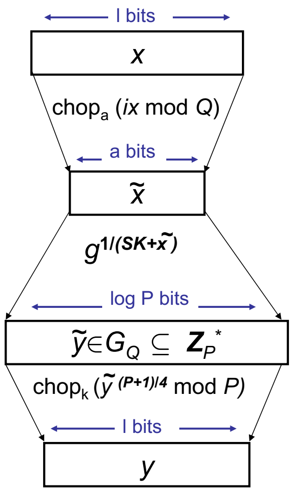

Fig. 1. Transformation of $F_{SK}^{DY}(\cdot)$ into a length-preserving PRF.

4 Distributed Pseudo-Random Permutations

We now show how to construct a **threshold pseudo-random permutation** by distributing the Luby-Rackoff construction. In principle, the Luby-Rackoff construction can be used with any PRF. However, we will use it with the PRF by Dodis-Yampolskiy [24], which allows us to evaluate the threshold PRP in only $O(1)$ communication rounds.

We begin by reviewing some formal definitions in Section 4.1. In Section 4.2, we show how to distribute a single Feistel permutation. In Section 4.3, we put all of the pieces together and explain how to distribute the entire Feistel cascade. Finally, in Section 4.4, we analyze our protocol's security and sketch how to make it proactive.

4.1 Definitions

Definition 2 (Feistel transformation). *Let $F : \{0,1\}^l \mapsto \{0,1\}^l$ be an l-bit mapping. We denote by $\bar{F}$ the permutation on $\{0,1\}^{2l}$ defined as $\bar{F}(x) = (x^R, x^L \oplus F(x^R))$, where $x = (x^L, x^R)$. Note that $\bar{F}$ is a permutation even if F is not. Its inverse is given by $\bar{F}^{-1}(y^L, y^R) = (f(y^L) \oplus y^R, y^L)$.*

Definition 3 (Feistel network). *Let $F_1, \ldots, F_k : \{0,1\}^l \mapsto \{0,1\}^l$ be l-bit mappings. Then a k-round Feistel network is a composition*

$$\Psi(F_1, \ldots, F_k) = \bar{F_1} \circ \bar{F_2} \cdots \bar{F_k}$$

Theorem 2 (Luby-Rackoff). *The permutation $\Psi(F_1, F_2, F_3, F_4)$ on $\{0,1\}^{2l}$ cannot be distinguished from a random permutation by a PPT adversary. Here, F_i are independently keyed pseudo-random functions.*

4.2 Distributed Feistel Transformation

In Section 3.2, we defined a PRF acting on $l(k) = \text{poly}(k)$ bit inputs by $F'_{i,SK}(x) = F^{DY}_{SK}(h_i(x))$. We also gave an $O(1)$ round protocol `PRF-DY` that computes shares of a PRF value $g^{1/(h_i(x)+SK)}$ from shares of input's bits and secret key. We now show how to distribute the Feistel transformation $\overline{F'_{i,SK}}$, which maps (x^L, x^R) to $(x^R, x^L \oplus F'_{i,SK}(x^R))$. The inverse Feistel transformation can be computed in a similar manner.

Our PRF protocol outputs shares of a random group element in G_Q. Meanwhile, we need a sharing of a random l-bit string to use in the Feistel transformation. We use a deterministic extractor, which does not lose any entropy, to extract l bits of randomness. In the centralized setting, given PRF output $\tilde{y} \in G_Q$, we can simply compute its square root by letting $y = (\tilde{y}^{(P+1)/4} \bmod P) \bmod 2^l$ (see also Figure 1). To distribute the extractor, we use a distributed exponentiation protocol followed by a conversion into bit shares. Notice that if we have shares of bits $x_i, y_i \in \{0,1\}$, denoted by $[x_i]^Q_j$ and $[y_i]^Q_j$, we can compute a share of their exclusive-OR as $[z_i]^Q_j \leftarrow [x_i]^Q_j + [y_i]^Q_j$ and $[z_i]^Q_j \leftarrow$ `MUL`$([z_i]^Q_j, 2 - [z_i]^Q_j \bmod Q)$.

Therefore, given bit shares of a $2l$-bit input (x^L, x^R), we can readily compute bit shares of a Feistel transformation:

Algorithm 3: One round of Feistel transformation $\texttt{FEISTEL}([i]_j^Q, [SK]_j^Q, [x_1]_j^Q, \ldots, [x_{2l}]_j^Q)$.

---

1 $[\tilde{y}]_j^P \leftarrow \texttt{PRF-DY}([i]_j^Q, [SK]_j^Q, [x_{l+1}]_j^Q, \ldots, [x_{2l}]_j^Q)$ ▷PRF value at x^R.
2 $[y]_j^P \leftarrow \texttt{EXP}_2([\tilde{y}]_j^P, (P+1)/4)$ ▷Extract square root $y^{(p+1)/4} \bmod P$.
3 $([y_1]_j^Q, \ldots, [y_l]_j^Q) \leftarrow \texttt{BITS}([y]_j^P, l, \mathbb{Z}_Q)$ ▷Truncate to l bits.
4 **for** $i \leftarrow 1$ **to** l *(in parallel)* ▷For all bits i
5 **do**
6 $\quad [z_i]_j^Q \leftarrow [x_i]_j^Q + [y_i]_j^Q \bmod Q$
7 $\quad [z_i]_j^Q \leftarrow \texttt{MUL}([z_i]_j^Q, 2 - [z_i]_j^Q \bmod Q)$ ▷We compute a share of $x_i \oplus y_i$.
8 **end**
9 **return** $([x_{l+1}]_j^Q, \ldots, [x_{2l}]_j^Q, [z_1]_j^Q, \ldots, [z_l]_j^Q)$ ▷Return shares of $(x^R, x^L \oplus F_{SK}^{DY}(x^R))$.

Security follows from composition theorem and security of its subprotocols. The protocol requires $O(1)$ rounds of communication between servers, because the for-loop is computed in parallel, and all other primitives take $O(1)$ rounds. The bit complexity is dominated by a call to the PRF-DY protocol in line 1 and by $l = o(m)$ calls to MUL in line 7, yielding $O((mn + m \log m)\mathcal{B})$ bit operations per player.

4.3 Distributed Luby-Rackoff Construction

Once we have a distributed protocol for the Feistel transformation, it is easy to distribute the Luby-Rackoff construction of PRP $g_s(x) = \Psi(F_1, F_2, F_3, F_4)(x)$. Initially, the n servers own shares of four independently chosen secret keys for the PRFs. These keys may either be jointly generated by servers or distributed to servers by a trusted party. An untrusted user, who wants to evaluate the PRP on input $x = (x^L, x^R)$, broadcasts x to the servers.[8] The servers convert x into bit shares and then run the distributed Feistel transformation for four rounds. We thus get:

[8] Alternatively, the user can split x into bit shares himself.

Algorithm 4: $\texttt{LUBY-RACKOFF}(([i_1]_j^Q, [SK_1]_j^Q), \ldots, ([i_4]_j^Q, [SK_4]_j^Q), x)$

$[0]_j^Q \leftarrow \texttt{JRPZ}(\mathbb{Z}_Q)$ ▷Shares of zero.
for $i \leftarrow 1$ **to** $2l$ *(in parallel)* ▷Locally compute shares of input's bits.
do
  $[y_i]_j^Q \leftarrow [0]_j^Q + x_i \bmod Q$
end
for $rnd \leftarrow 1$ **to** 4 ▷Run the Feistel transformation for four rounds.
do
  $([y_1]_j^Q, \ldots, [y_{2l}]_j^Q) \leftarrow \texttt{FEISTEL}([i_{rnd}]_j^Q, [SK_{rnd}]_j^Q, [y_1]_j^Q, \ldots, [y_{2l}]_j^Q)$
end
return $([y_1]_j^Q, \ldots, [y_{2l}]_j^Q)$

The round complexity is $O(1)$. Bit complexity is dominated by four calls to the Feistel protocol, which take $O((mn + m \log m)\mathcal{B})$ bit operations per player.

Similarly, we can distribute the inverse permutation $g_s^{-1}(\cdot)$ by replacing calls to Feistel transforms with calls to inverse Feistel transforms. We denote the resulting protocol by $\texttt{LUBY-RACKOFF}^{-1}$. The round and bit complexity remain the same.

4.4 Security

In the stand-alone case, the security of a PRP $g_s(\cdot) : \{0,1\}^{2l} \mapsto \{0,1\}^{2l}$ is formalized via a game between an attacker and an oracle. The attacker can query the oracle for $g_s(\cdot)$ and $g_s^{-1}(\cdot)$ on messages of his choice. Roughly, the PRP is deemed secure if no attacker can tell apart $g_s(x^*)$ from random for any message x^*, which was not asked as a query.

In the distributed setting, the attacker also gets transcripts of semi-honest servers. The **security property of threshold PRP** states that these transcripts do not help the attacker in any way. Formally, for any PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that breaks the security of threshold PRP by corrupting servers $P_{i_1}, \ldots, P_{i_\tau}$, there exists a PPT $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that breaks the security of the original PRP.

The attacker $\mathcal{A}$ learns key shares of corrupted servers. Then $\mathcal{A}_1$ runs in the first stage where it can interact with any honest servers on inputs of his choice. Attacker can ask servers either **encryption queries** where he learns shares of $g_s(x)$ or **decryption queries** for $g_s^{-1}(y)$. At the end of the phase, $\mathcal{A}_1$ outputs state information for $\mathcal{A}_2$ and a challenge input x^*, whose PRP value was not asked as a query. In the second stage, a random coin $b \in \{0,1\}$ is tossed. $\mathcal{A}_2$ receives a challenge Γ_b, which is either $\Gamma_0 \leftarrow g_s(x^*)$ or $\Gamma_1 \xleftarrow{\$} \{0,1\}^{2l}$. We let $\mathcal{A}_2$ interact with honest servers, but prohibit it from asking encryption queries on x^* or decryption queries on Γ_b. Finally, $\mathcal{A}_2$ outputs a guess b'. We say that $\mathcal{A}$ breaks the scheme if $\Pr[b = b'] > 1/2 + negl(k)$.

Theorem 3. $\texttt{LUBY-RACKOFF}$ *protocol is an* $\left\lfloor \frac{n-1}{2} \right\rfloor$*-secure threshold pseudo-random permutation in the static, honest-but-curious setting.*

Proof (sketch). In the honest-but-curious setting, `LUBY-RACKOFF` protocol correctly computes a permutation $g_s(x) = \Psi(F_1, F_2, F_3, F_4)(x)$ for some secret key $s = ((i_1, SK_1), \ldots, (i_4, SK_4))$. We thus concentrate on the pseudorandomness property.

For sake of contradiction, suppose there exists adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that breaks the security of `LUBY-RACKOFF`. Since $\mathcal{A}$ is static, we assume it corrupts the maximum allowed threshold of servers before the protocol starts[9]. By symmetry, we can assume corrupt servers P_j have indices $Bad = \{1, \ldots, \tau\}$. Bad servers learn their shares of secret key s. They also observe the protocol's input x, output $y = g_s(x)$, shares of output's bits $y_1, \ldots, y_{2l}$ of both good and bad servers, and all messages Ξ exchanged during the protocol. The adversarial view $\text{VIEW}_{\text{LUBY}, \mathcal{A}}$ is thus a random variable

$$\left\langle ([i_1]_k^Q, [SK_1]_k^Q), \ldots, ([i_4]_k^Q, [SK_4]_k^Q),\ x, y, [y_1]_j^Q, \ldots, [y_{2l}]_j^Q,\ \Xi \right\rangle$$

for $j = 1, \ldots, n$ and $k \in Bad$.

We construct a simulator $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that breaks the security of a PRP $g_s(\cdot)$. It will run $\mathcal{A}$ in a virtual distributed environment and imitate $\mathcal{A}$'s replies to distinguish $g_s(\cdot)$ from a random permutation, thereby violating Theorem 2.

Setup: Algorithm $\mathcal{B}$ generates random shares of keys for corrupt servers. For $j \in Bad$, it picks
$([i_1]_j^Q, [SK_1]_j^Q), \ldots, ([i_4]_j^Q, [SK_4]_j^Q) \xleftarrow{\$} \mathbb{Z}_Q^* \times \mathbb{Z}_Q^*$ and gives them to $\mathcal{A}$.

Responding to queries: When $\mathcal{A}$ initiates an honest server P_j ($j \notin Bad$) on input x, $\mathcal{B}$ in turn asks his oracle for $y = g_s(x)$. It generates random output shares $[z_1]_j^Q, \ldots, [z_{2l}]_j^Q \xleftarrow{\$} \mathbb{Z}_Q^*$ for $j \in Bad$. Then, $\mathcal{B}$ augments the set of shares of corrupted servers into a full and random sharing of y's bits. For each bit $y_i \in \{0, 1\}$ ($1 \le i \le 2l$), $\mathcal{B}$ picks a random polynomial $\alpha_i(x) \in \mathbb{Z}_Q[X]$ satisfying $\alpha_i(j) = [z_i]_j^Q$ and $\alpha_i(0) = y_i$. The adversary $\mathcal{A}$ receives randomized output shares $(\alpha_1(j), \ldots, \alpha_{2l}(j))$ for all servers P_j ($1 \le j \le n$). In the semi-honest setting, we can simulate the transcript of each subprotocol used by `LUBY-RACKOFF` given its input and output values. We can thus use these protocols as black-boxes and simulate messages Ξ in $\text{VIEW}_{\text{LUBY}, \mathcal{A}}$. These values provide a perfect simulation of the coalition's view. Decryption queries are handled just like encryption queries except $\mathcal{B}$ queries another oracle $g_s^{-1}(\cdot)$.

Challenge: Eventually, attacker $\mathcal{A}$ outputs a message x^* on which it wants to be challenged. It claims to be able to distinguish output of `LUBY-RACKOFF`(x^*) from a random $2l$-bit string. $\mathcal{B}$ sends the same challenge x^* to the trusted party and gets back Γ, which is either $g_s(x^*)$ or a random string. Finally, $\mathcal{B}$ gives Γ to $\mathcal{A}$.

Guess: Attacker $\mathcal{A}$ continues to issue queries for messages other than x^*. Simulator $\mathcal{B}$ responds to queries as before. Finally, $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$, which $\mathcal{B}$ also returns as its guess.

[9] If not, we can arbitrarily fix some of the honest servers to be corrupt.

$\square$

An adversary who controls less than $\tau = \lfloor (n-1)/2 \rfloor$ servers cannot break the privacy of our protocol. The protocol can easily be amended to achieve **proactive security** [31] and withstand the compromise of even all servers as long as at most τ servers are corrupted during each time period. The basic idea is to have servers periodically refresh their shares of the input and the secret keys. To be exact, each server P_j will from time to time execute the `JRPZ` protocol to generate a random share of zero, called $[0]_j^Q$. It will then update its input share to $[x]_j^Q \leftarrow [x]_j^Q + [0]_j^Q$ and its secret keys' shares to $[SK_i]_j^Q \leftarrow [SK_i]_j^Q + [0]_j^Q$.

5 Distributed Exponentiation

We describe how to distribute the computation of $x^y \bmod P$.

Earlier papers by Damgård *et al.* [15] and Shoup *et al.* [1] sketched how to implement distributed exponentiation for some of these scenarios. Unlike prior constructions, in our schemes the modulus P is publicly known rather than shared among the parties; this leads to simpler and more efficient protocols. We also allow the base and the exponent to be shared over different moduli P and Q.

5.1 Base x known, exponent y shared.

We can rewrite:

$$x^y = x^{\sum_{i=0}^{l-1} 2^i y_i} = \prod_{i=0}^{l-1} \left(y_i x^{2^i} + 1 - y_i \right) \bmod P.$$

Algorithm 5: Protocol $\mathtt{EXP}_1(x, [y]_j^Q)$: Base x is publicly known and exponent y is given as shares.

$([y_1]_j^P, \ldots, [y_l]_j^P) \leftarrow \mathtt{BITS}([y]_j^Q, \mathbb{Z}_P)$. ▷Obtain shares of exponent's bits.
for $i \leftarrow 1$ **to** l *(in parallel)* **do**
  $[z_i]_j^P \leftarrow [y_i]_j^P \cdot x^{2^{i-1}} + (1 - [y_i]_j^P) \bmod P$ ▷square-and-multiply algorithm used to compute x^{2^i}
end
$[z]_j^P \leftarrow \mathtt{MUL}([z_1]_j^P, \ldots, [z_l]_j^P)$ ▷Compute the product using
return $[z]_j^P$ ▷unbounded fan-in multiplication protocol.

The cost of the protocol is dominated by the bit conversion protocol. Its bit complexity is $O((mn + m \log m)\mathcal{B})$ and its round complexity is $O(1)$.

5.2 Base x shared, exponent y publicly known.

Damgård *et al.* [15] sketched how $x^y \bmod P$ can be computed in this scenario for any $x \neq 0$. Below, we give a detailed protocol implementing his idea.

Algorithm 6: $\texttt{EXP}_2([x]_j^P, y)$: Base x is given as shares, while exponent is publicly known.

---

2 $r_j \xleftarrow{\$} \mathbb{Z}_P^*$; $t_j \leftarrow r_j^y \bmod P$ ▷Choose a blinding factor r_j and r_j^y.

3 $\texttt{RANDSS}(r_j)$; $\texttt{RANDSS}(t_j)$ ▷Share r_j and t_j between players.

4 upon receiving $(([r_1]_j^P, \ldots, [r_n]_j^P) \wedge ([t_1]_j^P, \ldots, [t_n]_j^P))$:

5 $\quad [r]_j^P \leftarrow \texttt{MUL}([r_1]_j^P, \ldots, [r_n]_j^P)$ ▷A jointly generated random r

6 $\quad [t]_j^P \leftarrow \texttt{MUL}([t_1]_j^P, \ldots, [t_n]_j^P)$ ▷and $t = r^y$.

7 $\quad [z]_j^P \leftarrow \texttt{MUL}([x]_j^P, [r]_j^P)$ ▷This is a blinded share of xr,

8 $\quad$ **send**$([z]_j^P)$ to all players ▷which we can reveal.

9 upon receiving $([z]_1^P, \ldots, [z]_n^P)$:

10 $\quad z \leftarrow \sum_{j=1}^{\tau} \lambda_j [z]_j^P \bmod P$. ▷Interpolate shares to get $z = xr$.

11 $\quad w \leftarrow z^y \bmod P$

12 $\quad \texttt{RANDSS}(w)$ Share $w = (xr)^y$ between players.

13 **return** $\texttt{MUL}([w]_j^P, \texttt{INV}([t]_j^P))$ ▷A share of $z \cdot t^{-1} = (xr)^y \cdot (r^y)^{-1} = x^y \bmod P$.

This protocol is easily seen private as long as $x \neq 0$. It runs in $O(1)$ rounds. The running time is dominated by calls to multiplication protocol in lines 5-7 and by computation of Shamir secret sharing in lines 2-3. The bit complexity is thus $O(m^3 + n\mathcal{B})$.

5.3 Base x shared, exponent y shared.

Combining the above two protocols yields an exponentiation protocol where both the base and the exponent are shared. Essentially, the protocol is the same as $\texttt{EXP}_1$ except in line 5, we replace local multiplication with a call to distributed multiplication algorithm $\texttt{MUL}$. We also need to use $\texttt{EXP}_2$ to compute the shares of x^{2^i} for $i = 1, \ldots, l$. The final algorithm is as follows.

Algorithm 7: Protocol $\mathtt{EXP}([x]_j^P, [y]_j^Q)$: Both the base and the exponent are given as shares.

$([y_1]_j^P, \ldots, [y_l]_j^P) \leftarrow \mathtt{BITS}([y]_j^Q, \mathbb{Z}_P)$. ▷Obtain shares of exponent's $l = \lfloor \log Q \rfloor$ bits.
for $i \leftarrow 1$ **to** l *(in parallel)* **do**
  $[s]_j^P \leftarrow \mathtt{EXP}_2([x]_j^P, 2^{i-1} \bmod Q)$ ▷s is a share of x^{2^i}.
  $[z_i]_j^P \leftarrow \mathtt{MUL}([y_i]_j^P, [s]_j^P) + (1 - [y_i]_j^P) \bmod P$ ▷A share of $y_i x^{2^i} + (1 - y_i)$.
end
$[z]_j^P \leftarrow \mathtt{MUL}([z_1]_j^P, \ldots, [z_l]_j^P)$ ▷Compute the product using
return $[z]_j^P$ ▷unbounded fan-in multiplication protocol.

This protocol also runs in $O(1)$ rounds. The protocol performs l invocations of $\mathtt{EXP}_2$ and a single invocation of `BITS` and `MUL`; hence, it requires $O(m^4 + (mn + m \log m)\mathcal{B})$ bit operations per player.

6 Applications of Our Construction

In this paper, we have constructed a **threshold pseudorandom permutation**. Whenever a PRP is used as part of the construction, we can plug in our protocol instead.

Let $g_s : \{0,1\}^{2l} \mapsto \{0,1\}^{2l}$ be a $2l$-bit pseudo-random permutation obtained from the Luby-Rackoff construction $\Psi(F_1, F_2, F_3, F_4)$. The PRP's key s consists of four secret keys SK_i of pseudo-random functions F_i used in the construction. We denote by `LUBY-RACKOFF` our distributed protocol, which evaluates the $g_s(\cdot)$.

6.1 CCA-Secure Symmetric Encryption

A PRP is deterministic, so by itself it cannot be a secure encryption scheme [28]. The adversary can easily detect if the same message has been encrypted twice. Desai [18] described how a **CCA-secure symmetric encryption scheme** can be obtained from a PRP: The encryption $\mathcal{E}_s(m)$ of a $(2l-k)$-bit message m is defined as $\mathcal{E}_s(m) = g_s(m, r)$, where r is a k-bit randomly generated nonce. To decrypt, the user computes $\mathcal{D}_s(c) = g_s^{-1}(c)$ and extracts the message. To distribute the computation, the key s is split into shares among the n servers. Upon receiving a message m, the servers run the `JRP` protocol to generate shares of a secret random number r. They can extract shares of k bits, written $[r_1]_j^Q, \ldots, [r_k]_j^Q$, using the `BITS` protocol. Bit shares of m are easy to compute since m is public. Finally, the servers invoke `LUBY-RACKOFF` on $([m_1]_j^Q, \ldots, [m_{2l-k}]_j^Q, [r_1]_j^Q, \ldots, [r_k]_j^Q)$ to get shares of $g_s(m, r)$.

6.2 Authenticated Encryption

If we make the nonce r public and check during decryption that it matches the nonce in the ciphertext, then we get a distributed **authenticated encryption**

scheme (AE). Encryption of m is given by $\mathcal{AE}_s(m) = (r, g_s(m, r))$. The decryption algorithm $\mathcal{AD}_s(r', c)$ computes $(r, m) = g_s^{-1}(c)$ and checks that $r = r'$ before returning m to the user. The message here is rather short: It is limited to $(2l - k)$ bits by the length of the PRP. For longer messages, we can use an amplification paradigm of Dodis and An [23]: We compute a **concealment** (b, h) of message m $(|b| \ll |m|)$, which is a specialized publicly known transformation. In fact, we can even implement distributed **remotely keyed authenticated encryption** (RKAE) [7], where the servers do not need to perform any checks and just serve as PRP oracles for an untrusted user. The secret key s is split into shares among several computationally bounded **smartcards**, and an insecure, powerful **host** performs most of computations. The insecure host computes a concealment (b, h) of m and sends it to the smartcards, who run the `LUBY-RACKOFF` protocol, and return shares of $g_s(b)$.

6.3 Cipherblock Chaining Mode

We often need to encrypt messages that are longer than $2l$ bits. The message m is usually split into blocks $(m_1, \ldots, m_k)$ each of length $2l$. Then a PRP may be used in **cipher block chaining** (CBC) mode [35], which initializes c_0 with a random $2l$-bit string and sets $c_i = g_s(c_{i-1} \oplus m_i)$ for $i = 1, \ldots, k$. The encryption of m is defined to be $(c_0, c_1, \ldots, c_k)$. To decrypt, the user can compute $m_i = g_s^{-1}(c_i) \oplus c_{i-1}$. The servers own shares of secret key s. The untrusted user broadcasts message m to the servers. We must be careful to guard against the blockwise adaptive attacks [32]; hence, we require the user to send an entire message m. The servers run the `JRP` protocol to generate a random shared number from which shares of a $2l$-bit c_0 are extracted. For $i = 1, \ldots, k$ rounds, the servers distributively XOR shares of c_{i-1} and m_i (as in Section 4.3), and then run the `LUBY-RACKOFF` protocol on the result.

6.4 Variable Input Block Ciphers

Existing block ciphers operate on blocks of fixed length (FIL). Often, one needs a block cipher that can operate on inputs of variable length (VIL). There exist centralized constructions for VIL ciphers, which use a FIL block cipher as a black box: most notably, CMC [30], EME* [29] and an unbalanced Feistel network [39]. Our threshold PRP enables us to distribute the computation of these modes. Besides basic arithmetic operations, these modes XOR the ciphertexts (to distribute, we would use `BITS`), evaluate the fixed-length block cipher (`LUBY-RACKOFF`), compute the universal hash function $h_{a,b}(x) = ax + b$ (`MUL`), and truncate the outputs (`BITS`).

7 Conclusion

We gave a simple construction of a threshold PRP in the semi-honest model. Our scheme is fairly practical. PRPs are commonly used tools in protocol design. Our techniques enable distributing many protocols (using PRPs), which until now only existed in the centralized setting. In particular, we showed how to distribute the computation of a CBC encryption mode and a remotely keyed authenticated encryption scheme.

One open problem is whether we could use group multiplication to implement the distributed Feistel transformation rather than having to convert group elements into bit strings and avoid using the expensive protocol by Damgård *et al.* [16] altogether.

8 Acknowledgments

We thank Tomas Toft for his explanation of the bit conversion protocol.

References

1. J. Algesheimer, J. Camenisch, and V. Shoup. Efficient computation modulo a shared secret with applications to the generation of shared safe prime products. In *Advances in Cryptology - Proceedings of CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432. Springer-Verlag, 2002.
2. E. Bach. *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms*. A.C.M. Distinguished Dissertations. MIT press, Cambridge, MA, 1985.
3. J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds. In *Proceedings of the ACM Symposium on Principles of Distributed Computation*, pages 201–209, 1989.
4. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, pages 503–513, 1990.
5. M. Ben-or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 1–10, 1988.
6. E. Biham. A fast new DES implementation in software. In *Fast Software Encryption - Fourth International Workshop*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer-Verlag, 1997.
7. M. Blaze, J. Feigenbaum, and M. Naor. A formal treatment of remotely keyed encryption. In *Advances in Cryptology - Proceedings of EUROCRYPT 98*, Lecture Notes in Computer Science, pages 251–265. Springer-Verlag, 1998.
8. D. Boneh and M. K. Franklin. Efficient generation of shared RSA keys. *Journal of the Association for Computing Machinery*, 48(4):702–722, 2001.
9. E. F. Brickell, G. D. Crescenzo, and Y. Frankel. Sharing block ciphers. In E. Dawson, A. Clark, and C. Boyd, editors, *ACISP*, volume 1841 of *Lecture Notes in Computer Science*, pages 457–470. Springer, 2000.
10. C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology - Proceedings of CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer-Verlag, 2001.
11. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

12. D. Catalano, R. Gennaro, and S. Halevi. Computing inverses over a shared secret modulus. In *Advances in Cryptology - Proceedings of EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 190–206. Springer-Verlag, 2000.
13. D. Coppersmith, A. M. Odlyzko, and R. Schroeppel. Discrete logarithms in GF(p). *Algorithmica*, 1(1):1–15, 1986.
14. I. Damgård. Collision free hash functions and public key signature schemes. In *Advances in Cryptology - Proceedings of EUROCRYPT 87*, Lecture Notes in Computer Science, pages 203–216. Springer-Verlag, 1987.
15. 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 *Third Theory of Cryptography Conference*, 2006. To appear.
16. I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Advances in Cryptology - Proceedings of CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394. Springer-Verlag, 2005.
17. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Fourth International Workshop on Practice and Theory in Public Key Cryptography*, pages 119–136, 2001.
18. A. Desai. New paradigms for constructing symmetric encryption schemes secure against chosen-ciphertext attack. In *Advances in Cryptology - Proceedings of CRYPTO 2000*, pages 394–412, 2000.
19. Y. Desmedt. Society and group-oriented cryptography: a new concept. In *Advances in Cryptology - Proceedings of CRYPTO 87*, pages 120–127, 1987.
20. Y. Desmedt. Some recent research aspects of threshold cryptography. In *First International Workshop On Information Security*, pages 158–173, 1997.
21. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptology - Proceedings of CRYPTO 89*, pages 307–315, 1989.
22. Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In *Advances in Cryptology - Proceedings of CRYPTO 91*, pages 457–469, 1991.
23. Y. Dodis and J. H. An. Concealment and its applications to authenticated encryption. In *Advances in Cryptology - Proceedings of EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 312–329. Springer-Verlag, 2003.
24. Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *Eighth International Workshop on Theory and Practice in Public Key Cryptography*, pages 416–431, 2005.
25. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology - Proceedings of EUROCRYPT 99*, pages 295–310, 1999.
26. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. *Inf. Comput.*, 164(1):54–84, 2001.
27. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the Association for Computing Machinery*, 33:792–807, 1986.
28. S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the 14th Annual ACM Symposium on the Theory of Computing*, pages 270–299, 1982.
29. S. Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. In *Advances in Cryptology - Proceedings of INDOCRYPT 2004*, pages 315–327, 2004.
30. S. Halevi and P. Rogaway. A tweakable enciphering mode. In *Advances in Cryptology - Proceedings of CRYPTO 2003*, pages 482–499, 2003.

31. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in Cryptology - Proceedings of CRYPTO 95*, pages 339–352, 1995.
32. A. Joux, G. Martinet, and F. Valette. Blockwise-adaptive attackers. revisiting the (in)security of some provably secure encryption modes: CBC, GEM, IACBC. In *Advances in Cryptology - Proceedings of CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 17–30. Springer-Verlag, 2002.
33. M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal of Computing*, 17:373–386, 1988.
34. K. M. Martin, R. Safavi-Naini, H. Wang, and P. R. Wild. Distributing the encryption and decryption of a block cipher. *Designs, Codes, and Cryptography*, 2005. to appear.
35. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press LLC, Boca Raton, FL, 1997.
36. M. Naor, B. Pinkas, and O. Reingold. Distributed pseudo-random functions and KDCs. In *Advances in Cryptology - Proceedings of EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346. Springer-Verlag, 1999.
37. M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudorandom functions. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 458–467, 1997.
38. J. B. Nielsen. A threshold pseudorandom function construction and its applications. In *Advances in Cryptology - Proceedings of CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 401–416. Springer-Verlag, 2003.
39. S. Patel, Z. Ramzan, and G. S. Sundaram. Efficient constructions of variable-input-length block ciphers. In *Selected Areas in Cryptography 2004*, pages 326–340, 2004.
40. T. P. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology - Proceedings of EUROCRYPT 91*, pages 522–526, 1991.
41. T. Rabin. A simplified approach to threshold and proactive RSA. In *Advances in Cryptology - Proceedings of CRYPTO 98*, pages 89–104, 1998.
42. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the 21th Annual ACM Symposium on the Theory of Computing*, pages 73–85, 1989.
43. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
44. A. Yao. Protocols for secure computation (extended abstract). In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.