sustainability

MDPI

Article

Encapsulating Secrets Using Lockable Obfuscation and a RMERS-Based Public Key Encryption

Eduard Simion

Faculty of History, International Relations and Political Science, University of Oradea, Universitatii Street, 410087 Oradea, Romania; esimion@uoradea.ro

Abstract: Lockable obfuscation, a new primitive that occurs in cryptography, makes it possible to execute arbitrary polynomial-sized functions and recover a secret under specific equality conditions. More concretely, if the function executed over a specific input produces an output that matches an expected target value, here denoted by a, some secret string of bits s is exposed. Written in algebraic terms, if $f : \mathcal{X} \to \mathcal{A}$ has the property that for some $x, f(x) = a$, s is revealed. This work explores the possibility for safely decrypting ciphertexts, and based on the recovered plaintext's equality to a stored message, to reveal some secret. Concretely, this work provides a review of existing, well-known public key encryption schemes and argues for the efficiency of a new one relying on the ratio Mersenne hypothesis (RMERS), which is to be used in conjunction with a lockable obfuscator. This work explores the advantage conferred by this scheme, especially in the minimization of the branching program's number of levels that need to be obfuscated. The drawbacks of such schemes are also pointed out, given that they currently require the LWE evaluations level-per-level, one output bit at a time.

Keywords: lockable obfuscation; key encapsulation; Mersenne primes

check for updates

Citation: Simion, E. Encapsulating Secrets Using Lockable Obfuscation and a RMERS-Based Public Key Encryption. *Sustainability* **2022**, *14*, 11412. https://doi.org/10.3390/su141811412

Academic Editors: Debiao He, Qi Xie and Yuqi Dong

Received: 26 July 2022
Accepted: 7 September 2022
Published: 12 September 2022

1. Introduction

Various tasks require the manipulation of highly sensitive cryptographic data. In most of the cases, such use cases include securing communication channels, signing financial transactions, or distributing cryptographic keys that may be used for encryption (symmetric or, depending on the scenario, for public key decryption, digital signing, etc.). These data must be well-protected by the hardware or software implementations against all types of attacks.

On a current basis, such sensitive pieces of information, as the aforementioned ones, are provided in either hardware or software implementations. Considering the first case, such a secret piece of data can be easily destroyed through a physical process: for instance, using physically unclonable functions [1] (abbreviated PUFs), whenever the hardware will be investigated, through most of the known side-channel means, the PUF will internally destroy the protected key.

However, carrying hardware devices may be a cumbersome task, depending on the task. For instance, it may easily fell prey to routine airport checks. This makes us further consider the case of software. For instance, people carry smartphones or laptops as part of their daily routine, and it is less likely that existing software on such devices to be well-scrutinized during standard security checks, as an example, as compared to hardware.

Our work focuses on the problem of protecting such sensitive data in several scenarios that we elaborate below, in the motivational subsection. In such cases, the usage of public key encryption may be insufficient. Namely, it is desirable to prevent a secret from being shared between two parties in order to mitigate the human factor: for instance, where one party (A) knows that whenever party (B) sends the message "START THE ACTION", it will have to execute an operation. What is desirable to prevent are the risk factors: for instance, if A becomes corrupted or compromised, she may share this command with the adversary E. One step further ahead, E may encrypt the command and send it to A, pretending she is B. Thus, there may be need for authenticated encryption.

On the other hand, B may release a "black-box" software to A, which may indeed reveal such a command like "START THE ACTION", but only whenever the evaluation of some function passes. For instance, the black-box may encapsulate some decryption key *decK*, while B sends a ciphertext that is to be decrypted under this embedded *decK*; whenever the decryption process under *decK*, of the ciphertext *CT* sent by B produces some predefined value, such as "CORRECT", the secret information is then released.

Before discussing several key motivational aspects, this work elaborates more on the method of realizing such "black-boxes", from a mathematical, and then software-related point of view. Fortunately, there is no need to reinvent the wheel: at first sight, the existing range of cryptographic applications seem to match the need. A cryptographic mechanism, coined as virtual black-box obfuscation [2], is sufficient for this task. However, such virtual black-box obfuscation has been proven to be impossible for several simple tasks, and it may be generally more difficult to use it. However, the more recent literature propose the notion of lockable obfuscation [3,4], which is more akin to the simpler scenarios that are considered in this paper. Obfuscation itself has found practical examples in many areas of software engineering [5].

1.1. Motivational Aspects

Delegating trading capabilities: Wealthy individuals often delegate their savings designated for investments to professional traders; these traders are supposed to invest the financial assets as well as possible. In many circumstances, traders buy and sell stocks according to their own strategies, without asking for approval from the owners. However, things may change as the owners may want to give their approval for buying certain stocks of politically-sensitive companies, for instance. (See for instance the case of large telecom industry manufacturers.) In our somewhat related example, we consider an extremely volatile market—cryptocurrencies. Suppose a very busy but rich investor has already bought Ether, and stores the secret keys used to sign Ether transactions. Assume that Ether's price is relatively stable for around 1 year (a fact that has been confirmed by the trading record of this cryptocurrency), and the investor does not want to reveal his secret key to the trader until Ether grows. Once this is the case, for example, if the price of Ether grows 10 times in one week, the investor decides to sell and will hand in the key to the trader. Then, in a matter of hours (thus, over a relatively short period), the trader will buy and make profit on behalf of his investor.

The problem above is broken down into several components in order to analyze it: (*i*) the investor wants to be in full control of buying/selling stocks; (*ii*) the fact that stocks are very volatile prevents the investor from making the decision of buying within a very short time (i.e., minutes); (*iii*) the investor, once he decides he wants to invest in a specific stock, wants to delegate his rights to the trader; and (*iv*) the investor can "commit" to his secret key and will reveal it only at a specific moment, with the trader being authorized to sell in a couple of hours, for instance.

Signing keys usually look pseudorandom and are hard to memorize. So, they must be stored somewhere, which makes this procedure prone to cyber attacks. On top of that, instead of rushing and sending a stored secret key to the trader, utilizing an insecure channel, the key can be stored in a lockable obfuscator, and once the sender sends a randomly looking but specially designated message having some well-defined properties, the key is revealed.

As it can be easily observed from the use case present above, speed is an important criteria when working within such a scenario. It will be of no help for the trader if he learns the secret key that is used to transact a month after. Henceforth, we need to ensure the sensitive data are efficiently recoverable in a short amount of time.

Decrypt and Reveal: Several ultimate military scenarios have to deal with situations where a "letter of last resort" is required to fire an intercontinental ballistic nuclear missile. In such case, orders are communicated directly to the commanders of submarines carrying such annihilation weapons. Most of the time, a short alphanumeric code is transmitted and it has to be used to fire a missile. Several questions arise regarding the authenticity of such code: Can it be sent in plain? What happens if someone knows it at any time (such as the submarine commander or one of the officers)? Can the commander use this code at his will? Can it be reproduced in case the commander defects? Can an external adversary that knows it mimic the process of sending it and "fool" the submarine staff?

Even without any credible information on the topic, it is less likely that such messages are sent in plain. Encryption may be the standard and current technique for dealing with such cases. However, human players are always an issue: how can we deal with someone who knows such keys and who provides them to adversaries. In such a case, can an adversary deliberately send a cable to be allegedly decrypted to a nuclear launch order and code?

1.2. Discussion: Using Symmetric or Public Key Encryption within the Black-Box?

Although it makes a lot of sense to protect sensitive variables until they really need to be released, as seen in the motivational aspects presented above, a pressing question that comes to mind is the following: what sort of check should the "black-box" perform in order to release the secret? As a general rule, the more complex the function to be run in the black-box, the more time will by taken to perform such an operation.

A first option will be to use a symmetric encryption algorithm within the black-box. (From a theoretical point of view, a one-way function is sufficient here, as it provides a symmetric encryption scheme for free. However, we use a symmetric cipher as it is a more practical and popular concept to work with.) If this is the case, then the sender must store the key used to send messages to the black-box. In terms of our examples, the investor must store this symmetric key somewhere, then use it to encrypt messages to the black-box. So, instead of storing a random looking Ether key, he will now store a random looking symmetric key. Although the last one may be shorter, it is still hard to memorize.

The second question: Is public key encryption providing any benefit here? What if the investor uses a public key? He will need to send a message encrypted under the public key. The gain is that such a message may be memorable, and independent from the secret key and from the secretly stored key used to sign Ether transactions.

Henceforth, for the case of an investor, it makes more sense to use a public key encryption scheme within the obfuscated black-box construction.

1.3. Preamble: Public Key Encryption

Public key encryption (PKE) is a method that allows two parties to communicate in the presence of an adversary without pre-sharing any cryptographic secret. (Though, it must be noted that standard public key encryption schemes are prone to man-in-the-middle or botnet-based attacks [6]. For instance, an eavesdropper that owns the communication channel with A and B may simply interact with A pretending to be B, and interact with B, pretending they are A.) (Known as *key*.)

The history behind public key cryptography is by now half a century old, and started from ingenious mathematical ideas to a scientific branch, rigorously modeled using game theory, allowing for the modeling of attacks run by very strong opponents (attackers).

Early schemes. The early cryptographic schemes were inspired by the Diffie–Hellman key exchange. It should be noted, that although the simple Diffie–Hellman key exchange (that we describe in Section 3.4) was proposed in the 1970s, it was not until the middle of the 1980s that the first Diffie–Hellman inspired public key encryption scheme was proposed. The underlying hard problem was related to the discrete-log assumption. Meanwhile, the very different RSA cryptosystem had been proposed, based on a completely different assumption: factoring. Both schemes are extremely popular, and have been standardized and adopted by practitioners.

An interesting Mersenne assumption. Aggarwal, Joux, Prakash, and Santha [7] put forward a set of novel assumptions based on the properties of low Hamming weight elements sampled from some group of order p, where p is a Mersenne prime. The first of their assumptions states that the ratio of two elements over $\mathbb{Z}_p$ that have a low Hamming weight is pseudorandom, which we call the RMERS assumption.

The other is, which we call the Mersenne Low Hamming Weight Combination Assumption, is very similar to the Learning Parity with Noise problem (LPN): the adversary is required to distinguish tuples $(A, A \cdot S + E)$ from uniform ones, where S and E have Hamming weight h and A is a random elements over $\mathbb{Z}_p$. Based on those assumptions, the authors introduced an ingenious PKE scheme, for encrypting bit-by-bit, which is of great didactic importance through its simplicity. The practical performance of their first scheme is however, lame, as it encrypts a message bitwise, incurring ciphertexts proportional to the input length. A second scheme allows for the encryption of messages of multiple bits, through an error correcting code.

Those assumptions have been under scrutiny in the work of [8]. Later several other primitives have been developed based on those assumptions [9,10]. More recently, several primitives were built from related assumptions [11].

1.4. Preamble: Lockable Obfuscators

Obfuscation, as a computer science subject, deals with making programs unintelligible. Representing programs can be achieved in many forms, including PRAMs, Turing machines, or combinational logic circuits. These computational models are all equivalent in the runtime up to a polynomial factor.

Ideally, two obfuscated programs cannot be told apart, assuming they implement the same functionality. This intuition is captured formally within the notion of indistinguishability obfuscation [2].

While the notion of indistinguishability obfuscation is extremely complex, a more general notion exist, denoted virtual black-box obfuscation. This is impossible for all computable functions, but it turns out to be possible for something like point functions, or more recently, lockable functions [4].

A lockable function is defined as follows:

$$f_y : \mathcal{X} \to \mathcal{Y}\,,\ (y,s) \in \mathcal{Y}^2\,,\quad f_y(x) = \begin{cases} \perp\,, & \text{when } f_y(x) \neq y \\ s\,, & \text{when } f_y(x) = y \end{cases} \tag{1}$$

Several lockable obfuscators were proposed, and they are also suitable for storing decryption keys, as advertised by some of their use cases. These make them a prime candidate for the applications we use.

2. This Work

This work's contribution consists of a new, more efficient instantiation of a lockable obfuscation scheme, created on purpose for a specific public key encryption scheme working on Mersenne prime moduli.

As stated in Section 1, lockable obfuscation is a general paradigm that allows the user of an obfuscated (black-box) program to recover a secret value subject to the result of a computation given that $f(x) = \alpha$.

As may be hinted, this work needs the public key decryption to play the role of f, with the decryption key hidden in the obfuscator. The notion is guaranteed as long as y has sufficient min-entropy given the function f.

Thinking about software implementations and their performance, the decisive factor for constructing our lockable obfuscator is the branching program complexity of the decryption procedure.

The Main Observation

This paper analyses four of the publicly available and largely used standard public key encryption keys. First, several well-known candidates are quickly ruled out. During the process, the provided analysis is based on the textbook versions. (We avoid the PKCS standards because they have even worse efficiency and describing them does not serve to our purpose.) The main issue is the complexity of repeated modulo exponentiation of a large number.

A much simpler procedure is described in the LWE-based public key encryption scheme proposed by Regev. Although the scheme is inefficient as it encrypts bit-by-bit, it has an extremely simple decryption procedure, involving an inner product between the secret key and a ciphertext component.

However, the core observation made in this article is based on the decryption property of the Mersenne-based public key encryption scheme introduced in [7].

The main gain in working with Mersenne primes as moduli, namely $p = 2^n - 1$, with n itself a prime is the arithmetic: multiplying two elements modulo p induces a cyclic shift, in the following way:

$$a \cdot b := a \cdot (2^{b_0} + 2^{b_1} + \ldots + 2^{b_{\log(n)}}) \tag{2}$$

and each multiplication with a power of two induces the cyclic shift.

The obfuscator construction described in Section 5 is based on the reality that a decryption program is known at setup. We consider the relevant branching program, and encode the transition matrices that are used for program evaluation. There is a need to evaluate the input homomorphically over a ciphertext that encrypts the decryption key of the Mersenne-based public key. The interesting aspect is the branching program, which corresponds to the execution of one multiplication (up to $2n$ additions of n elements).

Thus being said, the scheme presented in this article is a custom-made lockable obfuscator. A pseudorandom generator is used to make sure that the decrypted value has sufficient min-entropy: instead of comparing a memorable value, we will compare a pseudorandom version of it with a random looking value.

3. Background Work

Notations related to algorithms: In this work, standard notations are used, and $\theta \in \mathbb{N}^*$ is considered as the security parameter of a an encryption algorithm (in close relation to its key length). An algorithm is implemented through a Turing machine, as a standard model of computations, and the inputs are given in their unary representation. PPT stands for "probabilistic polynomial-time", as a function of the security parameter. An important distinction must be made between a deterministic and a randomized algorithm.

Notations for the mathematical part: For an integer $q \geq 2$, we denote this by $\mathbb{Z}_q$ the ring of integers modulo q, and we represent it as $\mathbb{Z}_q = (-q/2, q/2]$. The "index" set $\{1, \ldots, k\}$ is represented as $[k]$. A real-valued function $\textsc{Negl}$ is negligible if $\textsc{Negl} \in \mathcal{O}(\theta^{-\omega(1)})$. We state that an event occurs with overwhelming probability if its probability is $1 - \textsc{Negl}$. The set of all negligible functions is written by $\textsc{Negl}$. A ordered list of n elements is written as $[a_1, \ldots, a_n]$.

Notations for Probability theory: Min-entropy and randomness extraction. The min-entropy of a random variable X is defined as $H_\infty(X) := -log_2(max_x Pr[X = x])$. Let $\mathrm{SD}(X, Y)$ denote the statistical distance between two random variables X and Y . This work sometimes refers to the Leftover Hash Lemma (LHL) from [12,13].

Notations related to adversaries: Given a randomized algorithm $\mathcal{A}$, we denote the action of running $\mathcal{A}$ on input(s) $(1^\theta, x_1, \ldots)$ over uniform randomness term r and assigning the output(s) to $(y_1, \ldots)$ by $\mathcal{A}(1^\theta, x_1, \ldots ; r) \to (y_1, \ldots)$. To simulate that $\mathcal{A}$ is given oracle access to some procedure $\mathcal{O}$, we write $\mathcal{A}^{\mathcal{O}}$. For any finite set S, we denote its cardinality by $|S|$. We sample an element x from the uniform distribution over S by $x \leftarrow_\$ S$. When another non-uniform distribution χ_S is used, we write $x \leftarrow_\chi S$.

3.1. Complexity Assumptions

The Learning With Errors Cryptographic Hypothesis. The Learning with Errors (LWE) (search version of the problem [14]) asks for the secret vector $\mathbf{s}$ over $\mathbb{F}_q^l$, given a polynomial-sized noisy vector of the form $\mathbf{A} \cdot \mathbf{s} + \mathbf{e}$, where $\mathbf{A}$ represents a randomly sampled matrix over $\mathbb{F}_q^{k \times \ell}$, and $\mathbf{e}$ represents a small error term sampled using the Normal distribution represented as χ. The decision version of the problem asks any PPT adversary to distinguish between the distribution of the LWE problem as opposed to the uniform distribution.

Ring-LWE. Lyubashevsky, Peikert, and Regev et al. [15] introduced the LWE version for polynomial rings. We assume that $R = \mathbb{Z}[x]/(x^n + 1)$ for n a power of 2, while $R_q := R/qR$, where q is a prime number constrained by $q = 1 \bmod 2n$.

Definition 1 (Ring LWE)**.** *Assume that s is sampled from* R_q *and denotes a secret. The adversary is given a polynomial number of samples that are all of the form* $(a, a \cdot s + e)$ *with a sampled from* R_q *and* $e \leftarrow_\chi R_q$*, (exclusive), or all uniformly sampled over* R_q^2*. The decision* $\mathsf{RLWE}_{q,\phi,\chi}$ *states that some* PPT*-bounded adversary cannot distinguish between the two settings with more than negligible advantage.*

Mersenne Low Hamming Weight Ratio Hypothesis.

Definition 2 (Mersenne Ratio Hypothesis)**.** *Let* $n \in \mathbb{N}^*$ *be a prime number and let* $p = 2^n - 1$ *be a prime number. Let* $h \in \mathbb{N}^*$ *and let* $W_h = \{x | x \in \{0,1\}^n \wedge \|x\|_2 = h\}$ *be the set of n bit integer having Hamming weight h. Let r be a random element over* $\mathbb{Z}_p$*. Let a and b be sampled uniformly at random from* W_h*. For any* PPT *adversary* $\mathcal{A}$*, its advantage in distinguishing between the following distributions:*

$$\left| \Pr[\mathcal{A}(1^n, (a{\cdot}b^{-1}) \bmod\ p) \to 1] - \Pr[\mathcal{A}(1^n, r) \to 1] \right|, \tag{3}$$

is negligible; all the operations are performed modulo p.

In the definition above, the adversary is given access to elements that are sampled either from the distribution $a \cdot b^{-1} \bmod p$ or from the uniform distribution, and it has to tell with sufficient advantage from which distribution the element has been sampled.

The decision variant of the Mersenne assumption is very similar to the Learning with Errors assumption.

3.2. Pairwise-Independent Hash Functions

This work assumes prior knowledge on the theory behind hash functions.

Definition 3. *Assume* $\mathcal{H}_\alpha$ *is a family of hash function. Each* $\mathsf{H} \in \mathcal{H}$ *has domain* $\{0,1\}^\ell$ *and range* $\{0,1\}^{\ell'}$*.* $\mathcal{H}$ *is pairwise independent if the following condition holds:*

$$\Pr[h(x) = a \wedge h(y) = b] = \frac{1}{2^{2\ell'}} \tag{4}$$

for all pairs input $(x, y) \in \{0,1\}^{2\ell}$ *and output pairs* $(a, b) \in \{0,1\}^{2\ell'}$*.*

3.3. Pseudorandom Number Generators

A pseudorandom number generator (PRNG, [16,17]) is in essence an algorithm that transforms a seed s, usually sampled from the uniform distribution into a (usually larger) output. The gist is that the output distribution should be indistinguishable to the uniform distribution over the co-domain.

Syntactically,

$$\mathsf{PRNG}(s) \to y, \forall s \in \mathcal{D}\,. \tag{5}$$

3.4. Public Key Encryption

Public key cryptography is the backbone used in implementing secure protocols over the Internet. Its roots originate in the public key exchange protocols proposed by Diffie and Hellman [18]. For the broad audience, we assume that Alice and Bob store each the secrets a and b; say both elements modulo some large prime p. Alice publishes g^a while Bob reveals g^b; suppose these two quantities are published online. Then, both Alice and Bob learn g^{ab} as $(g^a)^b$ modulo p for the case of Bob and $\left(g^b\right)^a$ modulo p for the case of Alice.

Once such a simple key exchange protocol was proposed, public key cryptography emerged. Researchers such as Taher El Gamal and Ronald Rivest, and Adi Shamir and Leonard Adleman observed that is easier to share secrets in a similar way to the exchange of the keys.

Definition 4 (Public key encryption)**.** *The following triple of algorithms represents a public key encryption scheme:*

- *A first step is meant to generate keys. The algorithm produces the public key* pk *and the secret key* sk*. We make the convention the secret key is n bits long and the public key has $P(n)$ bits, where P is a polynomial.*
- *The encryption routine encrypts the message M into the ciphertext CT using only the public key* pk*, without knowing the secret key* sk*; this routine is denoted by* Enc.
- *The decryption routine, can be seen as an inverse process with respect to the encryption routine: using the secret key* sk*, the message M is recovered from the ciphertext. This routine is denoted by* Dec.

The completeness of any public key encryption scheme is defined by the following equation:

$$\Pr[KeyGen(\alpha) \to (\mathsf{pk}, \mathsf{sk}) \wedge \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, M)) = M] \in 1 - \epsilon(\alpha) \tag{6}$$

where ϵ is the inverse of an exponential function.

The **security** of public key encryption is defined through game theory. The considered game includes an efficient adversary, and it interacts with an efficient benign entity:

Setup phase: The challenger chooses one well-formed pair of keys from the uniform distribution defined over the set of all possible key pairs. The public key is given to the adversary.

Challenge phase: The adversary chooses two messages of equal length, to be denoted as M_0 and M_1, and sends them to the challenger.

Encryption phase: One message out of the two, say message m_b, is selected and encrypted. Let the resulting ciphertext be CT_b, and let it be sent to the adversary.

Output phase: the adversary sends its output b', revealing if the message comes from m_0 or m_1.

The adversary wins the game, if:

$$\Pr\left[\text{Adversary}(\mathsf{pk}, CT_b, M_0, M_1) \to b' \wedge b' = b\right] = \frac{1}{2} + c \tag{7}$$

where c is a noticeable quantity.

3.5. Fully-Homomorphic Encryption

The notion of a fully homomorphic encryption [19–21] permits its users to evaluate some function (represented as a circuit) over a ciphertext. The result can be decrypted under the secret key paired with the encryption key. The decryption reveals the function applied over the original input.

Definition 5 (Fully homomorphic encryption)**.** *Given one function $f : \{0,1\}^n \to \{0,1\}$ with n bits as input, consider a circuit class $\mathcal{C}_{n,d}$ with input length n, depth d that implements f. The following set of algorithms represent a fully homomorphic encryption scheme:*

- *A first step is meant to generate keys. The corresponding algorithm produces the public key* pk *and the secret key* sk*. We make the convention that the secret key is n bits long and the public key has $P(n)$ bits, where P is a polynomial.*
- *The encryption routinely encrypts the message M into the ciphertext CT using only the public key* pk*, without knowing the secret key* sk*. Similarly to public key schemes, this step is denoted* Enc*.*
- *The evaluation routine* Eval *transforms any ciphertext CT into CT', by evaluating the circuit $C \in \mathcal{C}$ over CT, given access to the public key.*
- *The decryption routine can be seen as an inverse process with respect to the encryption routine: using the secret key* sk*, the message M is recovered from the ciphertext.*

The correctness and security conditions similar to a public key encryption scheme are not discussed here.

3.6. Leveled Homomorphic Encryption

A leveled fully homomorphic encryption scheme is in essence a fully homomorphic encryption scheme in a setting where the user is limited in the number of computations he will make, up to some level. It will not be described here, as it has the same syntax like fully homomorphic encryption. Hence, such a cryptographic tool is more restrictive.

3.7. Lockable Obfuscators

3.7.1. Intuition

The main purpose of a lockable obfuscator is to receive some value x as input, apply a (hidden) function f on x, and compare the result $f(x)$ with some pre-computed and stored values y. Whenever $f(x) = y$, a value z is revealed. As proposed, the value z can be a multi-input value, or a binary one. The security notion requires an obfuscated program to hide f and y, as long as y has sufficient min-entropy, given f.

In this work, the function f that is considered to be obfuscated can be represented as a circuit having logarithmic depth in its input length—namely, $f \in \mathsf{L/poly}$. We denote by BP its branching program representation, which has length L, input size ℓ_{in}, and output size ℓ_{out}; that is, $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$. In his well-known paper, David Barrington proves that every function with its circuit representation belonging to complexity class NC^1 has a branching program representation having its size bounded by a polynomial function in the input's size.

3.7.2. Formal Definition

The definition of a lockable obfuscator, introduced by Goyal, Koppula, and Waters in [3], is semantically equivalent to the notion computing and comparing obfuscators. The second notion has been introduced in [4], by Wichs and Zirdelis.

Definition 6 (Lockable obfuscation)**.** *Let $f : \{0,1\}^n \to \{0,1\}^m$ denote a function in NC^1 and let $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$. A lockable obfuscator LockObf with respect to (f,y) and inputs x consists of the following algorithms:*

LockObf.Setup$(1^\theta, (f,y,s))$**:** *takes as input the branching program representation of f and some value y. Returns the lockable obfuscator LockObf$[f,y]$.*

LockObf.Eval(x)**:** *given some input x, the LockObf$[f,y]$ computes $f(x)$. If $f(x) = y$, then return s. Otherwise, return $\perp$.*

The security definition is denoted by the term virtual black-box security.

An obfuscator obf for the distribution class $\mathcal{D}$ over a family of programs P satisfies distributional indistinguishability if there exists a (non-uniform) PPT *simulator $\mathcal{S}$, such that for every distribution ensemble $D = \{D_\theta\} \in \mathcal{D}$, we have*

$$(obf(1^\theta, P), aux) \approx_c (\mathcal{S}(1^\theta, P.prms), aux), \tag{8}$$

where $(P, aux) \leftarrow_\$ \mathcal{D}_\theta$.

As a side remark, although the works of [3,4] have the same operational interface, their constructions differ.

4. Efficiency of Decryption in Extant Public Key Schemes

Public key encryption schemes are defined in Section 3.4. As elaborated in the introductory part, when coupling their decryption circuit with lockable obfuscation, it is important to bear in mind that we want decryption to be as efficient as possible. This is due to the fact that lockable obfuscation adds an important computational overhead on top of the decryption circuit.

In this part, the decryption operation of some of the most well-established public key encryption schemes are briefly analyzed, and also some of the novel ones targeting efficiency.

4.1. The Textbook RSA Public Key Encryption Scheme

RSA [22] is one of the most commonly used public key encryption schemes. The algorithm has been patented in [23].

Key Generation: Sample two large and safe primes, p and q, and obtain their product N, which is denoted the RSA modulus. Sample an exponent e, such that $gcd(e, \phi(N)) = 1$. Publish (N, e) as the public key, while $d = e^{-1} \mod \phi(N)$ is the decryption key.

Encryption: to encrypt, compute $c = m^e \mod N$.

Decryption: to decrypt, compute $c^d \mod N$.

It is well known that the textbook version of RSA is not secure. This happens as encrypting the same message twice results in identical ciphertext, making the scheme prone to frequency attacks. However, the scheme has been standardized and randomness has been introduced during the encryption procedure in order to counter such simple attacks.

4.2. The El-Gamal Public Key Encryption Scheme

The El-Gamal [24] encryption scheme is extremely simple, close to the original Diffie–Hellman key exchange, is suitable for didactic purposes, and has a randomized encryption procedure (it is semantically secure).

Key Generation: Sample a safe prime $q = 2 \cdot p + 1$, where p is a prime and a generator g of order p. Sample a secret x over $\mathbb{Z}_p$ and publish g^x as the public key, while x is kept secret.

Encryption: To encrypt, sample a random element $r \in \mathbb{Z}_p$ and release the following pair $(g^r, g^{r \cdot x} \times M)$, where M is the message to be encrypted.

Decryption: The decryptor proceeds by (1) an exponentiation over the $\mathbb{Z}_p$, namely $g^{r \times x}$, as x is known by the decryptor while g^r is provided in the ciphertext; (2) an inversion that provides $(g^{r \times x})^{-1}$ and (3) a multiplication of $m \times g^{r \times x}$ with $(g^{r \times x})^{-1}$, which provides m.

As can be seen, the decryption operation provides three steps, which are all *computationally expensive*, as they involve exponentiation. The related standards of the scheme will not be detailed here, as it is beyond the purpose of this work.

4.3. Public Key Encryption from Learning with Errors

The Learning with Errors problem is recurrent in this work, as it forms the spinal chord of the lockable obfuscation scheme. Here, a simple public key encryption scheme described in [25] is exemplified.

Secret key: Choose a vector $\mathbf{s}$ uniformly at random over $\mathbb{Z}_q^n$. Set it as the secret key.

Public key: Sample m tuples $(\mathbf{a}_i, \mathbf{a}_i^t \cdot \mathbf{s} + e_i)$ from the Learning with Errors distribution under parameter regime q with Gaussian noise samples e_i.

Encryption: For each bit γ within the bit decomposition of the message, choose a random subset S in the powerset of $[m]$ and set the ciphertext to be:

$$(a, b) := \left(\sum_{i \in S} \mathbf{a}_i, \gamma \cdot [q/2] + \sum_{i \in S} b_i \right) \tag{9}$$

Decryption: Compute $b - \mathbf{a}^t \cdot \mathbf{s}$. If it is closer to 0 than $[q/2]$ return 0, otherwise 1.

It can be remarked that decryption is in fact a dot product multiplication of two vectors.

4.4. Public Key Encryption from the Mersenne Assumption

Remember the very simple way of encrypting a bit γ in [7]:

$$(-1)^{\gamma} \cdot \left(a \times \frac{c}{d} + b \right), \tag{10}$$

where all elements a, b, c, d over $\mathbb{Z}_p$ have a low Hamming weight. The public key consists of $\frac{c}{d}$ and the secret key is d. Decryption is performed by multiplying the ciphertext with d and determining whether the Hamming weight is small (plaintext is 0) or large (plaintext is 1).

Secret key: Chose a Mersenne prime moduli p of the form $2^h - 1$, where h is itself a prime number. Chose two elements c and d from the uniform distribution defined over $\mathbb{Z}_p$. Keep d secret.

Public key: Publish $\frac{c}{d}$.

Encryption: Select two elements, a and b from $\mathbb{Z}_p$. To encrypt a bit $\gamma \in \{0, 1\}$ release the ciphertext:

$$(-1)^{\gamma} \cdot \left(a \times \frac{c}{d} + b \right). \tag{11}$$

Decryption: Multiply $(-1)^{\gamma} \cdot \left(a \times \frac{c}{d} + b \right)$ with d and check whether the resulting value is below $2h^2$ (i.e., $\gamma = 0$) or above $p - 2h^2$ ($\gamma = 1$).

4.5. Discussion on Decryption Performance

Arguably, the two candidates that stand are the LWE-based encryption scheme, as well as the one based on the Mersenne assumption. This is because both RSA and ElGamal requires exponentiations mod N, which require repeated multiplications.

It is clear that the dot product between two vectors is a more complex operation to be supported, compared to a single multiplication of elements. The comparison parts count as the same.

Furthermore, the algebraic structure of the Mersenne assumption, namely the structure of the modulus p, makes it easier to implement, and this occurs faster on all software implementations that have been tested.

More concretely, for the envisioned obfuscation scheme considered, the structure of the branching program representing the function mattters the most. As a general rule, the lower the depth of the branching program, the more efficient the scheme.

5. Lockable Obfuscation for Mersenne-Based Public Key Encryption

The central result is presented in this section. To achieve it, we start from the lockable obfuscator presented in [3] (and concurrently by [4]) and adapt it to the decryption function we envision.

The most important aspect related to the scheme described in Section 4.4 is the arithmetic over Mersenne moduli. Namely, multiplying a and 2^z cyclically shifts the bits of x with z positions. To multiply and $a \cdot b$, it is easier to decompose b into powers of 2 and shift the bits of a accordingly, than perform the addition and output.

In this paper, consider a function f that can be represented as one circuit with a logarithmic depth in its input length—namely, an NC^1 circuit. We write by BP the corresponding branching program representation, which has length L, input size ℓ_{in}, and output size ℓ_{out}, that is, $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$.

5.1. Branching Program for Mersenne-Based PKE decryption

In this part, we figure out the depth of computing the decryption circuit for the Mersenne-based public key encryption scheme put forth in Section 4.4. It is clear that we will need to perform one multiplication and one comparison.

Luckily, the multiplication means shifting variables, and then performing an addition between the n elements. Clearly, the addition can be done in a tree-like structure, thus taking up to:

$$2 \cdot n \cdot \log_2(n) \tag{12}$$

steps, where $n = \log_2(p+1)$.

Clearly, such a program has logarithmic depth, making the size of the branching program polynom in the length of the input, using Barrington's result. The problem that we consider is to decrypt a ciphertext encrypting a memorable passphrase that gives the decryptor the secret s. However, we are constrained by the fact that the passphrase must have sufficient min-entropy. Instead of storing the passphrase, we hash it and then apply a pseudorandom number generator over the hash's output. The usage of the hash function is required since we need collision resistance. On the other hand, if we assume that the hash is pseudorandom, we will not need the PRNG. We have two man strategies. a) To build an obfuscator for the circuit performing decryption, hash, and PRNG, or to use leveled fully homomorphic encryption in conjunction with the universal circuit. We explore both cases and see that relying on the PKE in [7] is beneficial.

5.2. The Lockable Obfuscator for Mersenne-Based PKE

We provide a high-level overview and assume a black-box (read generic) utilization of a lockable obfuscator. Let Dec_d^i denote the decryption circuit outputting the i^{th} bit out of p for a [7] ciphertext. The lockable obfuscator has the following internal working:

Setup$(1^\alpha, M, s)$:

1. Let $\mathsf{Dec}_d(\cdot)$ be the circuit that computes the decryption in the scheme introduced in Section 4.4, where d is the secret key.
2. Choose a pairwise independent hash function, $\mathsf{H} \in \mathcal{H}$.
3. Choose a low depth pseudorandom generator PRNG.
4. Set $\mathsf{PRNG}(\mathsf{H}(M)) \to \beta \in \{0,1\}^\delta$.
5. Build a circuit C that computes the circuit. (The operations to be run are decryption, hash and PRG evaluation.):

$$C := \mathsf{PRNG} \circ \mathsf{H} \circ \mathsf{Dec}_d(\cdot)\ , \tag{13}$$

6. Sample FHE.*Setup*$(1^\theta) \to (\mathsf{pk}, \mathsf{sk})$, a pair of public private fully homomorphic encryption keys. Build a fully homomorphic ciphertext $\widetilde{CT}$ that encrypts the binary representation of the circuit C in Equation (15).

7. Build an obfuscator for the LWE decryption procedure:

$$LockObf.Setup(1^{\theta}, \mathsf{FHE.Dec}_{\mathsf{sk}}(\cdot), \mathsf{PRNG}(\mathsf{H}(M)), s) \rightarrow LockObf \tag{14}$$

Evaluation(M):

1. Construct the universal circuit $U_M(\cdot)$. A universal circuit is one that can evaluate any other circuit. In this case, we define $U_M(C(\cdot))$, for the circuit evaluating decryption over $(-1)^{\gamma} \cdot (a \cdot c/d + e)$.
2. Homomorphicaly evaluate $U_M()$ over the leveled homomorphic ciphertext $\widetilde{CT}$ and recovering one bit of the secret s if the M is correct.

We see here that the obfuscated circuit depends on the leveled homomorphic encryption scheme. We can compare the the depth of the circuit computing leveled homomorphic decryption with the combined depth of the circuit computing $\mathsf{PRNG}(\mathsf{H}(\mathsf{Dec}_d))$.

Depth of C**.** We can see that the depth of performing the FHE decryption is similar to an inner product, which has depth that is linear in the input length. Given that an inner product is to be achieved and the inner product involves a tree of $\log_2(n)$ additions, and each addition takes $\log_2(q)$ bits to perform multiplication, the branching program will have $2^{2\cdot(\log_2(n)+\log_2(q))}$ states.

The second construction. On the other hand, we consider the following scheme (high-level, as we abstract out the implementation of the *LockObf*).

Setup($1^{\alpha}, M, s$):

1. Let $\mathsf{Dec}_d(\cdot)$ be the circuit that computes the decryption in the scheme introduced in Section 4.4, where d is the secret key.
2. Choose a pairwise independent hash function, $\mathsf{H} \in \mathcal{H}$.
3. Choose a low depth pseudorandom generator PRNG.
4. Set $\mathsf{PRNG}(\mathsf{H}(M)) \rightarrow \beta \in \{0,1\}^{\delta}$.
5. Build a circuit C that computes the circuit (The operations to be run are decryption, hash, and PRG evaluation.):

$$C := \mathsf{PRNG} \circ \mathsf{H} \circ \mathsf{Dec}_d(\cdot)\ , \tag{15}$$

6. Build an obfuscator for the LWE decryption procedure (and also equality testing):

$$LockObf.Setup(1^{\theta}, \mathsf{PRNG}(\mathsf{Dec}_{\mathsf{sk}}(\cdot))), s) \rightarrow LockObf \tag{16}$$

Evaluation(M):

1. Evaluate the lockable obfuscator on input M and recover the secret bits of s.

5.3. Discussion

We obtain a lockable obfuscator for the decryption functionality within the Mersenne public key encryption scheme. If we get rid of the pairwise independent hash function ad rely only on the pseudo-randomness and (assumed) collision resistance of the PRNG, we can obtain a circuit with a very low depth. For instance, one can consider PRNGs with extremely low depth, such as PRNGs in NC^0. Thus, we can obtain a much shallower circuit and an even more reduced *LockObf* circuit.

Clearly, this second construction comes with a much shallower branching program, as the depth of the circuit is reduced compared to the previous one. It would be even more interesting to explore obtaining hash functions directly from the Mersenne Ratio assumption, and thus to further reduce the depth of the circuit to be obfuscated.

6. Conclusions

The current paper considers a problem of major importance for many practitioners: deploying sensitive data to potentially dishonest parties, and hiding them until a notification signaling a specific event is sent.

Given the questions about the receiver's honesty, that it shall not know the sensitive data a priori (this makes public key encryption alone useless), we have seen that employing lockable obfuscation is a solution to be considered.

This work studied the decryption's routine performance of several public key encryption schemes, and crafted a simplified lockable obfuscator by using a specific Mersenne Assumption-based public key encryption scheme.

Future Work

The current work can be extended in several directions. First, an implementation of the *LockObf* obfuscator with respect to the PKE scheme using the RMERS hypothesis can be provided. Second, the obfuscator can be further improved, in order to minimize its number of levels. Third, several other PKE schemes can be considered for implementation. Fourth, the same technique can be applied to different other primitives.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BP	Branching program
H	Hash function
LockObf	Lockable obfuscation
LWE	Learning with Errors assumption
NC^0	Nick's class 0
NC^1	Nick's class 1
PRNG	Pseudorandom number generator
PUF	Physical unclonable function
RMERS	Mersenne Ratio assumption
$(\mathbb{Z}_p^*, *)$	Multiplicative group of integers modulo prime p

References

1. Maes, R.; Verbauwhede, I. Physically unclonable functions: A study on the state of the art and future research directions. In *Towards Hardware-Intrinsic Security*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 3–37.
2. Barak, B.; Goldreich, O.; Impagliazzo, R.; Rudich, S.; Sahai, A.; Vadhan, S.P.; Yang, K. On the (Im)possibility of Obfuscating Programs. In *Advances in Cryptology—CRYPTO 2001, Proceedings of the 21st Annual International Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2001;* Lecture Notes in Computer Science; Kilian, J., Ed.; Springer: Heidelberg, Germany; Santa Barbara, CA, USA, 2001; Volume 2139, pp. 1–18. https://doi.org/10.1007/3-540-44647-8_1.
3. Goyal, R.; Koppula, V.; Waters, B. Lockable Obfuscation. In Proceedings of the 58th Annual Symposium on Foundations of Computer Science, Berkeley, CA, USA, 15–17 October 2017; Umans, C., Ed.; IEEE Computer Society Press: Berkeley, CA, USA, 2017; pp. 612–621. https://doi.org/10.1109/FOCS.2017.62.
4. Wichs, D.; Zirdelis, G. Obfuscating Compute-and-Compare Programs under LWE. In Proceedings of the 58th Annual Symposium on Foundations of Computer Science, Berkeley, CA, USA, 15–17 October 2017; Umans, C., Ed.; IEEE Computer Society Press: Berkeley, CA, USA, 2017; pp. 600–611. https://doi.org/10.1109/FOCS.2017.61.
5. Banescu, S.; Valenzuela, S.; Guggenmos, M.; Ahmadvand, M.; Pretschner, A. Dynamic Taint Analysis versus Obfuscated Self-Checking. In Proceedings of the Annual Computer Security Applications Conference, Virtual Event, 6–10 December 2021; Association for Computing Machinery: New York, NY, USA, 2021; p. 182–193. https://doi.org/10.1145/3485832.3485926.

6. Shetu, S.F.; Saifuzzaman, M.; Moon, N.N.; Nur, F.N. Survey of Botnet in Cyber Security. In Proceedings of the 2nd International Conference on Intelligent Communication and Computational Techniques (ICCT), Jaipur, India, 28–29 September 2019. https://doi.org/10.1109/icct46177.2019.8969048
7. Aggarwal, D.; Joux, A.; Prakash, A.; Santha, M. A New Public-Key Cryptosystem via Mersenne Numbers. In *Advances in Cryptology—CRYPTO 2018, Proceedings of the 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2018;* Part III; Lecture Notes in Computer Science; Shacham, H., Boldyreva, A., Eds.; Springer: Heidelberg, Germany; Santa Barbara, CA, USA, 2018; Volume 10993, pp. 459–482. https://doi.org/10.1007/978-3-319-96878-0_16.
8. Beunardeau, M.; Connolly, A.; Géraud, R.; Naccache, D. The Case for System Command Encryption. In Proceedings of the ASIACCS 17: 12th ACM Symposium on Information, Computer and Communications Security, Abu Dhabi, United Arab Emirates, 2–6 April 2017; Karri, R., Sinanoglu, O., Sadeghi, A.R., Yi, X., Eds.; ACM Press: Abu Dhabi, United Arab Emirates, 2017; p. 6.
9. Ferradi, H.; Xagawa, K. Post-quantum Provably-Secure Authentication and MAC from Mersenne Primes. In *Topics in Cryptology—CT-RSA 2020, Proceedings of the The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, 24–28 February 2020*; Lecture Notes in Computer Science; Jarecki, S., Ed.; Springer: Heidelberg, Germany; San Francisco, CA, USA, 2020; Volume 12006, pp. 469–495. https://doi.org/10.1007/978-3-030-40186-3_20.
10. Ferradi, H.; Naccache, D. Integer Reconstruction Public-Key Encryption. In *Cryptology and Network Security, Proceedings of the 18th International Conference, CANS 2019, Fuzhou, China, 25–27 October 2019*; Lecture Notes in Computer Science; Mu, Y., Deng, R.H., Huang, X., Eds.; Springer: Heidelberg, Germany; Fuzhou, China, 2019; Volume 11829, pp. 412–433. https://doi.org/10.1007/978-3-030-31578-8_23.
11. Das, D.; Joux, A.; Narayanan, A.K. Fiat-Shamir Signatures without Aborts Using Ring-and-Noise Assumptions. Cryptology ePrint Archive, Paper 2022/205. 2022. Available online: https://eprint.iacr.org/2022/205 (accessed on 12 June 2022).
12. Håstad, J.; Impagliazzo, R.; Levin, L.A.; Luby, M. A pseudorandom generator from any one-way function. *SIAM J. Comput.* **1999**, *28*, 1364–1396.
13. Dodis, Y.; Reyzin, L.; Smith, A. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, 2–6 May 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 523–540.
14. Regev, O. On lattices, learning with errors, random linear codes, and cryptography. In Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, 22–24 May 2005; Gabow, H.N., Fagin, R., Eds.; ACM Press: Baltimore, MA, USA, 2005; pp. 84–93. https://doi.org/10.1145/1060590.1060603.
15. Lyubashevsky, V.; Peikert, C.; Regev, O. On Ideal Lattices and Learning with Errors over Rings. In *Advances in Cryptology—EUROCRYPT 2010, Proceedings of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, 30 May–3 June 2010*; Lecture Notes in Computer Science; Gilbert, H., Ed.; Springer: Heidelberg, Germany; French Riviera, France, 2010; Volume 6110, pp. 1–23. https://doi.org/10.1007/978-3-642-13190-5_1.
16. Håstad, J.; Impagliazzo, R.; Levin, L.A.; Luby, M. Construction of a pseudo-random generator from any one-way function. *SIAM J. Comput.* **1993**, *28*, 12–24.
17. Luby, M.; Rackoff, C. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.* **1988**, *17*, 373–386.
18. Diffie, W.; Hellman, M. New directions in cryptography. *IEEE Trans. Inf. Theory* **1976**, *22*, 644–654.
19. Gentry, C. Fully homomorphic encryption using ideal lattices. In Proceedings of the 41st Annual ACM Symposium on Theory of Computing, Bethesda, MD, USA, 31 May–2 June 2009; Mitzenmacher, M., Ed.; ACM Press: Bethesda, MD, USA, 2009; pp. 169–178. https://doi.org/10.1145/1536414.1536440.
20. Gentry, C.; Sahai, A.; Waters, B. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Advances in Cryptology—CRYPTO 2013, Proceedings of the 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2013*; Part I; Lecture Notes in Computer Science; Canetti, R.; Garay, J.A., Eds.; Springer: Heidelberg, Germany; Santa Barbara, CA, USA, 2013; Volume 8042, pp. 75–92. https://doi.org/10.1007/978-3-642-40041-4_5.
21. Coron, J.S.; Mandal, A.; Naccache, D.; Tibouchi, M. Fully Homomorphic Encryption over the Integers with Shorter Public Keys. In *Advances in Cryptology—CRYPTO 2011, Proceedings of the 31st Annual Cryptology Conference, Santa Barbara, CA, USA, 14–18 August 2011*; Lecture Notes in Computer Science; Rogaway, P., Ed.; Springer: Heidelberg, Germany; Santa Barbara, CA, USA, 2011; Volume 6841, pp. 487–504. https://doi.org/10.1007/978-3-642-22792-9_28.
22. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126.
23. Rivest, R.L.; Shamir, A.; Adleman, L.M. Cryptographic Communications System and Method. US Patent 4,405,829, 20 September 1983.
24. ElGamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory* **1985**, *31*, 469–472.
25. Regev, O. The learning with errors problem. *Invit. Surv. CCC* **2010**, *7*, 11.