Cryptography with Updates

Prabhanjan Ananth[1⋆], Aloni Cohen[2⋆⋆], and Abhishek Jain[3⋆⋆⋆]

[1] UCLA
prabhanjan@cs.ucla.edu
[2] MIT
aloni@mit.edu
[3] Johns Hopkins University
abhishek@cs.jhu.edu

Abstract. Starting with the work of Bellare, Goldreich and Goldwasser [CRYPTO'94], a rich line of work has studied the design of updatable cryptographic primitives. For example, in an updatable signature scheme, it is possible to efficiently transform a signature over a message into a signature over a related message without recomputing a fresh signature.

In this work, we continue this line of research, and perform a systematic study of updatable cryptography. We take a unified approach towards adding updatability features to recently studied cryptographic objects such as attribute-based encryption, functional encryption, witness encryption, indistinguishability obfuscation, and many others that support non-interactive computation over inputs. We, in fact, go further and extend our approach to classical protocols such as zero-knowledge proofs and secure multiparty computation.

To accomplish this goal, we introduce a new notion of *updatable randomized encodings* that extends the standard notion of randomized encodings to incorporate updatability features. We show that updatable randomized encodings can be used to generically transform cryptographic primitives to their updatable counterparts.

We provide various definitions and constructions of updatable randomized encodings based on varying assumptions, ranging from one-way functions to compact functional encryption.

⋆ Work done in part while visiting the Simons Institute for Theoretical Computer Science, supported by the Simons Foundation and and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467. This work was partially supported by grant #360584 from the Simons Foundation.

⋆⋆ Supported in part by NSF MACS CNS-1413920, DARPA IBM W911NF-15-C-0236, and Simons Investigator Award Agreement Dated 6-5-12

⋆⋆⋆ Work done in part while visiting the Simons Institute for Theoretical Computer Science, supported by the Simons Foundation and and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467. Supported in part by a DARPA/ARL Safeware Grant W911NF-15-C-0213 and NSF CNS-1414023.

1 Introduction

The last decade has seen the advent of a vast array of advanced cryptographic primitives such as attribute-based encryption [55, 45], predicate encryption [20, 57, 47, 43], fully homomorphic encryption [36], fully homomorphic signatures [7, 17, 44], functional encryption [55, 19, 54, 41], constrained pseudorandom functions [21, 22, 48], witness encryption [34, 38], witness PRFs [60], indistinguishability obfuscation [9, 32], and many more. Most of these primitives can be viewed as "cryptographic circuit compilers" where a circuit C can be compiled into an encoding $\langle C \rangle$ and an input x can be encoded as $\langle x \rangle$ such that they can be evaluated together to compute $C(x)$. For example, in a functional encryption scheme, circuit compilation corresponds to the key generation process whereas input encoding corresponds to encryption. Over the recent years, cryptographic circuit compilers have revolutionized cryptography by providing non-interactive means of computing over inputs/data.

A fundamental limitation of these circuit compilers is that they only support *static* compilation. That is, once a circuit is compiled, it can no longer be modified. In reality, however, compiled circuits may need to undergo several updates over a period of time. For example, consider an organization where each employee is issued a decryption key SK_P of an attribute-based encryption scheme where the predicate P corresponds to her access level determined by her employment status. However, if her employment status later changes, then we would want to update the predicate P associated with her decryption key. Known schemes, unfortunately, do not support this ability.

Motivated by the necessity of supporting updates in applications, in this work, we study and build *dynamic* circuit compilers. In a dynamic circuit compiler, it is possible to update a compiled circuit $\langle C \rangle$ into another compiled circuit $\langle C' \rangle$ by using an *encoded update string* whose size only depends on the "difference" between the plaintext circuits C and C'. For example, if the difference between C and C' is simply a single gate change, then this should be reflected in the size of the encoded update. Note that this rules out the trivial solution of simply releasing a new compiled circuit at the time of update.

Background: Incremental Cryptography. The study of cryptography with updates was initiated by Bellare, Goldreich and Goldwasser [10] under the umbrella of *incremental cryptography*. They studied the problem of incremental digital signatures, where given a signature of a message m, it should be possible to efficiently compute a signature of a related message m', without having to recompute the signature of m' from scratch. Following their work, the study of incremental cryptography was extended to other basic cryptographic primitives such as encryption and hash functions [10, 11, 52, 31, 12, 24, 53], and more recently, indistinguishability obfuscation [35, 5].

Our Goal. In this work, we continue this line of research, and perform a systematic study of updatable cryptographic primitives. We take a unified approach towards adding updatability features to recently studied primitives such

as attribute-based encryption, functional encryption and more generally, cryptographic circuit compilers. We, in fact, go further and also study updatability for classical protocols such as zero-knowledge proofs and secure multiparty computation.

To accomplish this goal, we introduce a new notion of *updatable randomized encodings* that extends the standard notion of randomized encoding [46] to incorporate updatability features. We show that updatable randomized encodings can be used to generically transform cryptographic primitives (discussed above) to their updatable counterparts.

Updatable Randomized Encodings. The notion of randomized encoding [46] allows one to encode a "complex" computation $C(x)$ into a "simple" randomized function $\mathsf{Encode}(C, x; r)$ such that given the output $\langle C(x) \rangle$ of the latter, it is possible to recover the value $C(x)$ (by running a public Decode algorithm) but it is impossible to learn anything else about C or x. The typical measure of complexity studied in the literature is parallel-time complexity or circuit depth. Such randomized encodings are known to exist for general circuits based on only the existence of one-way functions [6] (also referred to as Yao's garbled circuits [59], where $\mathsf{Encode}(C, x; r)$ is in $\mathbf{NC}^1$).

In this work, we study *updatable* randomized encodings (URE): given a randomized encoding $\langle C(x) \rangle$ of $C(x)$, we want the ability to update it to an encoding $\langle C'(x') \rangle$ of $C'(x')$, where C' and x' are derived from C and x by applying some "update" $\mathbf{u}$. For now, we may think of this update as some small modification to the circuit or input (e.g., change the output gate of C to AND and the second bit of x to 1). We require that the update $\mathbf{u}$ can be encoded as $\langle \mathbf{u} \rangle$ which can then be used to transform $\langle C(x) \rangle$ into $\langle C'(x') \rangle$, a randomized encoding of $C'(x')$. A bit more precisely, a URE scheme consists of the following algorithms:

- $\mathsf{Encode}(C, x)$ takes as input a circuit C and an input x, and outputs an encoding $\langle C(x) \rangle$ and a secret state st.
- $\mathsf{GenUpd}(\mathsf{st}, \mathbf{u})$ takes as input an update $\mathbf{u}$, and outputs an encoded update $\langle \mathbf{u} \rangle$ and a possibly updated state st'.
- $\mathsf{ApplyUpd}\left(\langle C(x) \rangle, \langle \mathbf{u} \rangle\right)$ takes as input a randomized encoding $\langle C(x) \rangle$ and an update encoding $\langle \mathbf{u} \rangle$, and outputs an (updated) encoding $\langle C'(x') \rangle$.
- $\mathsf{Decode}\left(\langle C(x) \rangle\right)$ takes as input a (possibly updated) randomized encoding $\langle C(x) \rangle$, and outputs the value $y = C(x)$.

If we make no additional requirements, the above could be easily achieved. For instance, let Encode output the state $\mathsf{st} = (C, x)$, and let GenUpd – which now has access to C and x from st in addition to the update $\mathbf{u}$ – compute the updated C' and x' directly and output a as the encoded update $\langle \mathbf{u} \rangle$ the standard randomized encoding of $\langle C'(x') \rangle$. $\mathsf{ApplyUpd}$ would correspondingly output $\langle \mathbf{u} \rangle = \langle C'(x') \rangle$. The drawback of this approach is that a fresh randomized encoding is computed during every evaluation of GenUpd, irrespective of whether $\mathbf{u}$ constitutes a minute or significant change to the underlying C and x.

Our key efficiency requirement is that the running time of the GenUpd algorithm must be a fixed polynomial size of the update (and a security parameter),

and independent of the size of the circuit and input being updated. This, in particular, implies that the size of an update encoding $\langle \mathbf{u} \rangle$ is also a fixed polynomial in the size of $\mathbf{u}$ (and the security parameter).

The above discussion immediately generalizes to the setting of *multiple sequential updates*.[4] Let $\langle C_0(x_0) \rangle$ denote an initial randomized encoding. Let $\mathbf{u}_1, \ldots, \mathbf{u}_n$ denote a sequence of updates and let $\langle \mathbf{u}_i \rangle$ denote an encoding of $\mathbf{u}_i$. In a URE scheme for multiple updates, $\langle C_0(x_0) \rangle$ can be updated to $\langle C_1(x_1) \rangle$ using $\langle u_1 \rangle$; the result can then be updated into $\langle C_2(x_2) \rangle$ using $\langle u_2 \rangle$, and so on, until we obtain $\langle C_n(x_n) \rangle$. We allow the number of updates n to be an arbitrary polynomial in the security parameter.

Within this framework, two distinct notions naturally arise.

URE with multiple evaluations: Every intermediate encoding $\langle C_i(x_i \rangle$ can be decoded to obtain $C_i(x_i)$. For security, we require that given an initial randomized encoding $\langle C_0(x_0) \rangle$ and a sequence of encoded updates $\{\langle \mathbf{u}_i \rangle\}_{i=1}^n$, an adversary can learn only the outputs $\{C_i(x_i)\}_{i=0}^n$, and nothing else.

URE with single evaluation: Only the final encoding $\langle C_n(x_n) \rangle$ can be decoded. To enable this, we will consider an augmented decoding algorithm that additionally requires an "unlocking key."[5] This unlocking key is provided after all the updates are completed, allowing the user to decode the final encoding, but preventing her from decoding any intermediate values. For security, we require that given an initial randomized encoding $\langle C_0(x_0) \rangle$ and a sequence of encoded updates $\{\langle \mathbf{u}_i \rangle\}_{i=1}^n$), an adversary can only learn the final output $C_n(x_n)$, and nothing else.

Except where otherwise specified, we use URE to mean the multiple-evaluation variant. For both conceptual reasons and to minimize confusion, we in fact consider an alternative but equivalent formulation of single-evaluation URE which we call **updatable garbled circuits** (UGC). A garbled circuit [59] is a "decomposable" randomized encoding, where a circuit C and an input x can be encoded separately. In an updatable garbled circuit scheme, given an encoding $\langle C_0 \rangle$ of a circuit C_0 and a sequence of update encodings $\langle \mathbf{u}_1 \rangle, \ldots, \langle \mathbf{u}_n \rangle$, it is possible to compute updated circuit encodings $\langle C_1 \rangle, \ldots, \langle C_n \rangle$, where C_i is derived from C_{i-1} using $\mathbf{u}_i$. Once all the updates are completed, an encoding $\langle x \rangle$ for an input x is released. This input encoding can then be used to decode the final circuit encoding $\langle C_n \rangle$ and learn $C_n(x_n)$. Intuitively, the input encoding can be viewed as the unlocking key in single-evaluation URE.

It is easy to see that UGC is a weaker notion than multi-evaluation URE. In particular, since UGC only allows for decoding "at the end," it remains *single-use*, while multi-evaluation URE captures *reusability*.

[4] One may also consider an alternative notion of *parallel* updates, where every update $\langle \mathbf{u}_i \rangle$ is applied to the *original* encoding $\langle C_0(x_0) \rangle$. It turns out that URE with parallel updates is closely connected to the notion of reusable garbled circuits[42]. We refer the reader to the full version [2] for further discussion on this subject.

[5] In the setting of bounded updates, this modification is unnecessary. We focus primarily on the unbounded setting.

We find the notions of URE and UGC to be of interest from a purely complexity-theoretic perspective. Further, as we discuss later, they have powerful applications to updatable cryptography.

1.1 Our Results

In this work, we initiate the study of updatable randomized encodings. We study both simulation and indistinguishability-based security definitions and obtain general positive results. We showcase URE as a central object for the study of updatable cryptography by demonstrating applications to other updatable cryptographic primitives. The technical ideas we develop for our constructions are quite general, and may be applicable to future works on updatable cryptography.

Multi-evaluation URE for General Updates. Before stating our positive results for multi-evaluation URE, we first informally describe which classes of updates we can support. An update $\mathsf{Update} \in \mathcal{U}$ represents some way to modify any circuit C and an input x to some modified circuit C' and input x'. We denote by $\mathbf{u}$ the procedure $(C', x') \leftarrow \mathbf{u}(C, x, \mathsf{Update})$ which applies the update to C and x. We consider all $\mathcal{U}$ and Update subject to two restrictions: (1) Update is computed by a (family) of circuits, one for every circuit size $|C|$, and (2) Update preserves circuit size (i.e., $|C| = |C'|$). We refer to this very broad class of updates as *general circuit updates*.

For general circuit updates, we construct URE from *compact* functional encryption. The summary below focuses on indistinguishability-based security, and concludes with a remark on achieving simulation-based security.

Theorem 1 (Informal). *Assuming the existence of secret-key, compact functional encryption supporting a single key query and B ciphertexts, there exists a multi-evaluation* URE *scheme supporting B sequential general circuit updates.*

A compact functional encryption is one where the running time of the encryption algorithm for a message m is a fixed polynomial in the size of m and the security parameter, and independent of the complexity of the function family supported by the FE scheme.

For the case of **unbounded** updates, a recent work of Bitansky et al. [14] shows that secret-key compact functional encryption with unbounded-many ciphertexts implies exponentially-efficient indistinguishability obfuscation (XIO) [49]. Put together with the results of [49] and [3, 15], it shows that *sub-exponentially* secure secret-key compact FE that supports a single function key query together with the learning with errors (LWE) assumption implies indistinguishability obfuscation.

In contrast, in Theorem 1, we require secret-key compact FE with only *polynomial security*. Such an FE scheme can be based on polynomial-hardness assumptions on multilinear maps using the results of [33] and [15, 4].

For the case of **polynomially-bounded** updates, we can, in fact, relax our assumption to only *one-way functions*. We obtain this result by using a *stateful* single-key compact secret-key FE scheme for an a priori bounded number B

of ciphertexts. A stateful single-key compact secret-key FE scheme can be constructed from garbled circuits: a functional key consists of B garbled circuits, i^{th} ciphertext consists of garbled wire keys corresponding to the i^{th} garbled circuit. This FE scheme is stateful since the encryption algorithm needs to store how many messages it has encrypted so far.

Plugging in such an FE scheme in Theorem 1 yields the following corollary.

Corollary 1 (Informal). *Assuming one-way functions, for any fixed polynomial B, there exists a multi-evaluation* URE *scheme supporting B sequential general circuit updates.*

On the necessity of functional encryption. It is natural to ask whether secret-key compact FE is necessary for building multi-evaluation URE with unbounded updates. We show that if a (multi-evaluation) URE scheme is *output compact*, then it implies XIO. Put together with the result of [49], we have that a URE scheme with output compactness together with LWE implies a public-key compact FE scheme that supports a single key query.

Theorem 2 (Informal). *Assuming LWE, a multi-evaluation* URE *scheme with unbounded output-compact updates implies a public-key compact FE scheme that supports a single key query.*

In an output-compact URE scheme, the running time of the GenUpd algorithm is independent of the output length of the updated circuit. We remark that the URE scheme obtained from Theorem 1 is, in fact, output compact. Our construction in Theorem 1 is in this sense tight.

On output compactness. We study both indistinguishability and simulation-based security notions for URE. In the context of FE, it is known from [1, 26] that simulation-secure FE with output compactness is impossible for general functions. We observe that the same ideas as in [1, 26] can be used to establish impossibility of simulation-secure URE with output compact updates.

However, when we consider indistinguishability-based security, URE with output compact updates is indeed possible. The results in Theorem 1 and Corollary 1 are stated for this case. Furthermore, using the trapdoor circuits technique of [26], one can generically transform output-compact URE with indistinguishability security to non-output-compact URE with simulation-based security.

Updatable garbled circuits with gate-wise updates. We now turn to updatable garbled circuits, an alternate formulation of single-evaluation URE. We consider the family of gate-wise updates, where an update $\mathbf{u}$ can modify a single gate of a circuit or add or delete a gate. Below, we consider the case of unbounded updates and bounded updates separately.

UGC with Unbounded Updates from Lattice Assumptions. Our first result is a construction of UGC for general circuits that supports an unbounded number of sequential updates from the family of gate-wise updates. We build such a scheme from worst-case lattice assumptions.

Theorem 3 (Informal). *Let $\mathcal{C}$ be a family of general circuits. Assuming the hardness of approximating either GapSVP or SIVP to within sub-exponential factors, there exists a* UGC *scheme for $\mathcal{C}$ that supports an unbounded polynomial number of sequential gate-wise updates.*

We defer the proof of this theorem to the full version. [2] At the heart of this result is a new notion of *puncturable symmetric proxy re-encryption scheme* that extends the well-studied notion of proxy re-encryption [16]. In a symmetric proxy re-encryption scheme, for any pair of secret keys SK_1, SK_2, it is possible to construct a re-encryption key $\mathsf{RK}_{1\to 2}$ that can be used to publicly transform a ciphertext w.r.t. SK_1 into a ciphertext w.r.t. SK_2. In our new notion of puncturable proxy re-encryption, re-encryption keys can be "disabled" on ciphertexts CT^* (w.r.t. SK_1) s.t. the semantic security of CT^* holds even if the adversary is given the punctured key $\mathsf{RK}_{1\to 2}^{\mathsf{CT}^*}$ and SK_2. We give a construction of such a scheme based on the hardness of approximating either GapSVP or SIVP to within sub-exponential factors.

Given the wide applications of proxy re-encryption (see, e.g., [8] for a discussion), we believe that our notion of puncturable proxy re-encryption is of independent interest and likely to find new applications in the future.

UGC with Bounded Updates from One-Way Functions. For the case of a polynomially-bounded number of updates, we can relax our assumption to only *one-way functions*. We obtain this result by using a puncturable PRF scheme that can be based on one-way functions [40, 56].

Theorem 4 (Informal). *Let $\mathcal{C}$ be a family of general circuits, and λ be a security parameter. Assuming one-way functions, for any fixed polynomial p, there exists a* UGC *scheme for $\mathcal{C}$ that supports $p(\lambda)$ sequential gate-wise updates. The size of the initial garbled circuit as well as each update encoding is independent of p. However, the initial circuit garbling time and update generation time grows with p.*

The construction of this scheme is quite simple and does not require a puncturable proxy re-encryption scheme. We provide an informal description of this scheme in the technical overview section 2.1.

Applications. We next discuss applications of our results.

Updatable Primitives with IND security. We start by discussing application of multi-evaluation URE to dynamic circuit compilers. Here, we demonstrate our main idea by a concrete example, namely, by showing how to use URE to transform any (key-policy) attribute-based encryption (ABE) scheme into *updatable ABE*. The same idea can be used in a generic way to build dynamic circuit compilers and obtain updatable functional encryption, updatable indistinguishability obfuscation, and so on. We refer the reader to the full version [2] for the general case.

We briefly describe a generic transformation from any ABE scheme to one where the policies associated with secret keys can be updated. The setup and

encryption algorithms for the updatable ABE scheme are the same as in the underlying ABE scheme. The key generation algorithm in the updatable ABE scheme works as follows: to compute an attribute key for a function f, we compute a URE $\langle C_f \rangle$ of a circuit C_f where C runs the key generation algorithm of the underlying ABE scheme using function f and outputs a key SK_f. To decrypt a ciphertext, a user can first decode $\langle C_f \rangle$ to compute SK_f and then use it to decrypt the ciphertext.

In order to update an attribute key for a function f to another key for function f', we can simply issue an update encoding $\langle \mathbf{u} \rangle$ for $\langle C_f \rangle$ where $\mathbf{u}$ captures the modification from f to f'. To compute the updated attribute key, a user can first update $\langle C_f \rangle$ using $\langle \mathbf{u} \rangle$ to obtain $\langle C_{f'} \rangle$, and then decode it to obtain an attribute key $\mathsf{SK}_{f'}$ for f'.

Let us inspect the efficiency of updates in the above updatable ABE scheme. As in URE, we would like the size (as well as the generation time) of an update encoding here to be independent of the size of the updated function. Note, however, that the output of the updated function $C_{f'}$ is very large – an entire attribute key $\mathsf{SK}_{f'}$! Thus, in order to achieve the aforementioned efficiency, we require that the URE scheme has updates with output compactness.

Recall that URE with output compact updates is only possible with indistinguishability based security. As such, the above idea is only applicable to cryptographic primitives with indistinguishability-based security.

Updatable Primitives with SIM security. Next, we discuss applications of URE to cryptographic primitives with simulation-based security. In the main body of the paper, we describe two concrete applications, namely, *updatable non-interactive zero-knowledge proofs* (UNIZK) and *updatable multiparty computation* (UMPC). A notable feature of these constructions is that they only require a URE scheme with *non-output-compact* updates and simulation-based security. Below, we briefly describe our main idea for constructing UNIZKs.

Let (x, w) denote an instance and witness pair for an **NP** language L. Let $\mathbf{u}$ denote an update that transforms (x, w) to another valid instance and witness pair (x', w'). In a UNIZK proof system for L, it should be possible for a prover to efficiently compute an encoding $\langle \mathbf{u} \rangle$ of $\mathbf{u}$ that allows a verifier to transform a valid proof π for x into a proof π' for x' and verify its correctness.

We now briefly describe our transformation. A proof π for (x, w) in the UNIZK scheme is computed as follows: we first compute a URE $\langle C_{x,w} \rangle$ for a circuit $C_{x,w}$ that checks whether (x, w) satisfies the **NP** relation associated with L and outputs 1 or 0 accordingly. Furthermore, we also compute a regular NIZK proof ϕ to prove that $\langle C_{x,w} \rangle$ is computed "honestly." To verify $\pi = (\langle C_{x,w} \rangle, \phi)$, a verifier first verifies ϕ and if the check succeeds, it decodes $\langle C_{x,w} \rangle$ and outputs its answer.

In order to update a proof π, we can simply issue an update encoding $\langle \mathbf{u} \rangle$ for the randomized encoding $\langle C_{x,w} \rangle$, along with a regular NIZK proof ϕ' that $\langle \mathbf{u} \rangle$ was computed honestly. Upon receiving the update $(\langle \mathbf{u} \rangle, \phi')$, a verifier can first verify ϕ' and then update $\langle C_{x,w} \rangle$ using $\langle \mathbf{u} \rangle$ to obtain $\langle C_{x',w'} \rangle$. Finally, it

can decode the updated URE $\langle C_{x',w'} \rangle$ to learn whether x' is in the language L or not.

It should be easy to see that the above idea can, in fact, be also used to make *interactive* zero-knowledge proofs updatable. Finally, we note that the above is a slightly oversimplified description and we refer the reader to the full version [2] for further details on UNIZK and UMPC, respectively.

1.2 Related Work

Incremental Cryptography. The area of incremental cryptography was pioneered by Bellare, Goldreich and Goldwasser [10]. While their work dealt with signature schemes, the concept of incremental updates has been subsequently studied for other basic cryptographic primitives such as hash functions, semantically-secure encryption and deterministic encryption [11, 52, 31, 24, 53]. To the best of our knowledge, all of these works only consider bit-wise updates, in which a single bit of the message is modified.

While our work shares much in spirit with these works, we highlight one important difference. In incremental cryptography, update operation is performed "in house," e.g., in the case of signatures, the entity who produces the original signature also performs the update. In contrast, we consider a *client-server* scenario where the client simply produces an update encoding, and the actual updating process is performed by the server. This difference stipulates different efficiency and security requirements. On the one hand, incremental cryptography necessarily requires efficient updating time for the notion to be non-trivial, while we consider the weaker property of efficient update encoding generation time. On the other hand, our security definition is necessarily stronger since we allow the adversary to view the update encodings – a property not necessary when the updating is done "in house."

Incremental/Patchable Obfuscation. Recently, [35] and [5] study the notion of updatability in the context of *indistinguishability obfuscation*. The work of [35] considers incremental (i.e., bit-wise) updates, while [5] allow for arbitrary updates, including those that may increase the size of the program (modeled as a Turing machine).

We note that one of our results, namely, URE with unbounded updates can be derived from [5] at the cost of requiring sub-exponentially secure iO. In contrast, we obtain our result by using *polynomially secure* secret-key compact FE.

Malleable NIZKs. Our notion of updatable NIZKs should be contrasted with the notion of malleable NIZKs proposed by Chase et al. [28]. In a malleable NIZK, it is possible to publicly "maul" a proof string π for a statement x into a a proof string π' for a related statement x'. In contrast, our notion of UNIZK only allows for privately generated updates. To the best of our knowledge, malleable NIZKs are only known either for a limited class of update relations from standard assumptions [28], or for general class of update relations based on non-falsifiable assumptions such as succinct non-interactive arguments [29]. In contrast, we

show how to build UNIZK for unbounded number of general updates from compact secret-key FE and regular NIZKs, and for a bounded number of general updates from regular NIZKs.

Updatable Codes. The concept of updating was also studied in the context of error correcting codes by [27]. In this context, it is difficult to model the problem of updating – we should be able to change few bits of the code to correspond to a codeword of a different message and at the same time we want the distance between codewords of different messages to be far apart. We refer the reader to their work for discussion on this seemingly contradictory requirement. In a subsequent work, [30] studied this problem in the context of non-malleable codes.

2 Our Techniques

We start with the construction of UGC and present the main ideas underlying the construction. We then build upon the intuition developed in the construction of UGC, to construct (multi-evaluation) URE.

2.1 Construction of UGC

Below, we restrict our discussion to updates that correspond to a gate change.

A *Lock-and-Release* Mechanism for Single Update. Let us first start with the simpler goal of building a UGC scheme that supports updating a *single* gate. Let C be a circuit comprised of s-many gates $C^1,\ldots,C^s$. Our starting idea towards a UGC scheme with single update is as follows: in order to garble C, we simply compute a garbling of C using a standard gate-by-gate garbling scheme such as [59].[6] We denote by $\langle C\rangle_{\mathsf{gc}}$ the garbled circuit for C, and by $\langle C\rangle^i_{\mathsf{gc}}$ the garbled gate corresponding to gate C^i. Encrypt each garbled gate, and output the resulting ciphertexts $\mathsf{CT}_1,\ldots,\mathsf{CT}_s$.

Now, suppose we wish to update garbling of C to garbling of C' where C' only differs from C in the first gate. Then, a natural idea is to release a decryption key that only decrypts the ciphertexts $\mathsf{CT}_2,\ldots,\mathsf{CT}_s$ (but not CT_1). The encoding of the update is consists of these $s-1$ keys and, along with a garbled-version of the new gate $\langle C'\rangle^1_{\mathsf{gc}}$. Using this information, the receiver can decrypt and recover the garbled gates $\langle C\rangle^2_{\mathsf{gc}},\ldots,\langle C\rangle^s_{\mathsf{gc}}$. Together with $\langle C\rangle^1_{\mathsf{gc}}$, this forms a complete garbled circuit for C'.

The main remaining question is how to implement the aforementioned *conditional decryption* mechanism. A naive way to achieve this is to encrypt each ciphertext with an independent encryption key, and then release the decryption key for every position $i \neq 1$. However, note that in this naive solution, the size

[6] In gate-by-gate garbling schemes such as [59], each boolean gate can be garbled knowing only the circuit topology and the gate's functionality, independently of the remainder of the circuit.

of the update encoding is proportional to $s = |C|$. In terms of size, this is no better than garbling C' from scratch.

To address this, we could instead use a (secret key) *puncturable* encryption scheme. In such a scheme, for any ciphertext CT, it is possible to compute "punctured decryption key" that enables one to decrypt all ciphertexts except CT. In order to be non-trivial, the size of punctured decryption keys must be independent of the number of ciphertexts generated. Such an encryption scheme can be built from puncturable pseudorandom functions [56, 22, 21, 48] (c.f. Waters [58]) which in turn can be based on any one-way function. It is easy to verify that given such an encryption scheme, we can efficiency we desire in above construction for UGC supporting a single update.

We find it instructive to abstract the above idea as a **lock-and-release** mechanism. Roughly speaking, the encryption of the wire keys corresponding to C constitutes the *locking* step, while the dissemination of the punctured decryption key constitutes the (conditional) *release* step. We find this abstraction particularly useful going forward, in order to develop our full solution for an unbounded number of updates.

Multiple Updates: Main Challenges. The above solution does not offer any security for multiple sequential updates – even for two. If the two updates for two different gates would allow an adversary to recover a garbling of the original circuit Additionally, the above scheme does not "connect" the two updates in any manner; an adversary could choose to apply none, one, or both of the updates before evaluating the circuit.

A *Layered Lock-and-Release* Mechanism for Bounded Updates. We next consider the case of an a priori *bounded* number of updates (the setting of Theorem 4). The key idea, in a nutshell, is to use *layered* punctured encryption, or alternatively, a **layered lock-and-release** mechanism.

Suppose we wish to handle p-many of updates. When garbling the circuit C, instead of encrypting the garbled gates a single time, we instead use p "onion" layers of encryption scheme, each using a punctured encryption scheme. Let $\mathbf{u}_1,\ldots,\mathbf{u}_p$ be a sequence of gate updates, each consisting of a gate $g \in [s]$ to change and a new gate type. To generate an updatable garbled circuit for C, first garble $\tilde{C}$ using a traditional gate-by-gate scheme. Sample p keys $SK_1,\ldots,SK_p$ for a puncturable encryption scheme. Encrypt each garbled gate $\langle C\rangle^i_{\mathsf{gc}}$ of the garbled circuit in p layers, yielding a ciphertext $\mathsf{CT}_i = \mathsf{Enc}(SK_1, \mathsf{Enc}(SK_2, \ldots \mathsf{Enc}(SK_p, \langle C\rangle^i_{\mathsf{gc}})))$.

The encoding of the first update $\mathbf{u}_1 = (g_1, \mathsf{gateType}_1)$ simply corresponds to releasing a decryption key for the outermost encryption layer that is punctured at CT_{g_1}, along with a layer $(p-1)$ encryption $\mathsf{CT}'_{g_i} = \mathsf{Enc}(SK_2, \mathsf{Enc}(SK_3, \ldots \mathsf{Enc}(SK_p, \langle C'\rangle^{g_i}_{\mathsf{gc}})))$, where $\langle C'\rangle^{g_i}_{\mathsf{gc}}$ is the new garbled gate corresponding to replacing gate g_i of C with $\mathsf{gateType}$. Likewise, an encoding of the i-th update $\mathbf{u}_i$ corresponds to releasing a punctured decryption key for SK_i, $(i-1)$ encryption of the new garbled gate.

The above idea of layered (punctured) encryption ensures that the receiver cannot "skip" any update, and instead must apply all the updates one-by-one to "peel-off" all the encryption layers from the garbled gates. Furthermore, since the encryption layers can only be removed in a prescribed order, the receiver must applies the updates *in order*. Finally, after all the decryption operations, the receiver only obtains a single garbled gate every location in the (updated) circuit.

We now briefly argue that the above construction satisfies the efficiency properties stated in Theorem 4. We first note that punctured encryption scheme in the above construction can simply correspond to a one-time pad where the randomness for computing the ith ciphertext, for every $i \in |C|$, is generated by evaluating a puncturable PRF over the index i. The PRF key (i.e., the secret key for the punctured encryption scheme) is different for every layer. With this instantiation, note that the size of the initial garbled circuit as well as every update is independent of the total number of updates p; however, the garbling time as well as update generation time depends on p.

A *Relock-and-Eventual-Release* Mechanism for Unbounded Updates. The above solution is that it inherently requires the number of updates to be a priori bounded. To support multiple updates, our main insight is to develop a **relock-and-eventual-release** mechanism as opposed to the layered lock-and-release mechanism discussed above. That is, instead of removing a lock at every step, our idea is to *change the lock* at every step. In encryption terminology, our idea is to replace the layered encryption in the above approach with a symmetric *re-encryption* scheme [16]. In a symmetric re-encryption scheme, given two encryption keys SK_1 and SK_2, it is possible to issue a re-encryption key $\mathsf{RK}_{1\to 2}$ that transforms any ciphertext w.r.t. SK_1 into a ciphertext w.r.t. SK_2. In order to allow for updates, we, require the re-encryption scheme to support key puncturing. That is, we require that it is possible to compute a punctured re-encryption key $\mathsf{RK}^{\mathsf{CT}^*}_{1\to 2}$ that allows one to transform any ciphertext w.r.t. SK_1 into a ciphertext w.r.t. SK_2, *except the ciphertext* CT^* (computed under SK_1). From a security viewpoint, we require that the semantic security of CT^* should hold even if the adversary is given $\mathsf{RK}^{\mathsf{CT}^*}_{1\to 2}$ *and the terminal secret key* SK_2. We refer to such an encryption scheme as a puncturable symmetric re-encryption scheme. While the above description only refers to a "single-hop" puncturable re-encryption scheme, we in fact consider a "multi-hop" scheme.

Armed with the above insight, we modify the previous solution template as follows: the garbling of a circuit C consists of $\tilde{U}$ as before. The main difference is that the wire keys $w_C = \{w_{C_1}, \ldots, w_{C_n}\}$ corresponding to the circuit C are now encrypted w.r.t. a puncturable re-encryption scheme. Let SK_0 denote the secret key used to encrypt the wire keys. In order to issue an update encoding for an update $\mathbf{u}_i$, we release (a) a re-encryption key $\mathsf{RK}^{\mathsf{CT}}_{i-1\to i}$ that is punctured at ciphertext CT, where CT is the encryption of w_{C_ℓ} w.r.t. SK_{i-1} and ℓ is the position associated with update $\mathbf{u}_i$, along with (b) an encryption of $w_{\bar{C}_\ell}$ w.r.t. SK_i. For the final update L, we simply release the Lth secret key SK_L.

We argue the security of the construction by using the security of the puncturable re-encryption scheme and the garbling scheme (see the technical sections for details). We note, however, that this construction does not hide the location of the updates. Indeed, the correctness of the above scheme requires the evaluator to know the locations that are being updated. To address this, we provide a generic transformation from any UGC scheme (or in fact, any URE scheme) that does not achieve update hiding into one that achieves update hiding. Our transformation uses non-interactive oblivious RAM in the same manner as in [35]. Finally, we note that while the above only discusses single-bit updates, our construction handles multi-bit updates as well.

The only missing piece in the above solution is a construction of a puncturable symmetric re-encryption scheme. We discuss it next.

Puncturable Symmetric Re-encryption from Worst-case Lattice Assumptions. The work of [18] constructs re-encryption schemes from *key homomorphic PRFs,* which have the property that for all x, K_1, and K_2, $\mathsf{PRF}(K_1, x) + \mathsf{PRF}(K_2, x) = \mathsf{PRF}(K_1 + K_2, x)$, where the keys and outputs of the PRF lie in appropriate groups. A secret key for the encryption scheme is simply a PRF key, and the encryption of a message m with secret key K_1 and randomness r is $\mathsf{CT} = (r, m + \mathsf{PRF}(K_1, r))$.

A re-encryption key between between secret keys K_1 and K_2 is simply their difference: $\mathsf{RK}_{1\to 2} = K_2 - K_1$. The key-homomorphism suggests a natural way to re-encrypt ciphertexts, as $(r, m + \mathsf{PRF}(K_1, r) + \mathsf{PRF}(\mathsf{RK}_{1\to 2}, r)) = (r, m + (K_2, r))$ is a ciphertext w.r.t K_2. Observe that successful re-encryption of a ciphertext with randomness r relies on the ability to compute $\mathsf{PRF}(\mathsf{RK}_{1\to 2}, r)$.

We construct *puncturable* proxy re-encryption scheme following the above approach, but instantiated with *constrained* key-homomorphic PRFs [23]. A punctured re-encryption key $\mathsf{RK}_{1\to 2}^{\mathsf{CT}^*}$ for a ciphertext CT^* with randomness r^* is the PRF key $K_2 - K_1$ punctured at the input r^*. This key, which can be used to evaluate $\mathsf{PRF}(K_2 - K_1, r)$ for all $r \neq r^*$, enables the re-encryption of all ciphertexts except for the ciphertext CT^*.

For security, we require that the semantic security of CT^* holds given both $\mathsf{RK}_{1\to 2}^{\mathsf{CT}^*}$ and K_2. We reduce to the security of the constrained PRF, which guarantees that $y^* := \mathsf{PRF}(K_2 - K_1, r^*)$ is pseudorandom. The key idea is that (partial information about) y^* can be computed given CT^*, K_2, and (partial information about) the message m.

2.2 Construction of URE

We now shift our focus on building multi-evaluation URE.

Relock-and-Release Mechanism. Recall that the main difference between UGC and URE is that UGC only allows for a single evaluation after a sequence of updates, while URE allows for evaluation after *every* update. As such, the relock-and-*eventual*-release mechanism that we discussed above does not suffice for

building URE. Our starting insight is to instead develop a **relock-and-release** mechanism that performs both relocking and release at every step. Intuitively, relocking allows us to "carry over" the updates, while the release mechanism allows us to evaluate the updated randomized encoding at every step.

Starting Idea: Garbled RAM with Persistent Memory. A natural starting approach to implement such a relock-and-release mechanism is via the use of garbled RAMs with persistent memory [51, 37]. In a garbled RAM scheme, it is possible to encode a database D_0 and later issue encodings for RAM programs $M_1, \ldots, M_n$. Each RAM program encoding $\widetilde{M}_i$ updates the database encoding from $\widetilde{D_{i-1}}$ to $\widetilde{D_i}$, and outputs the result of some computation on D_i.

Given this description, it is not difficult to see why such a notion is useful for our purpose. Starting from a garbled RAM scheme and a standard randomized encodings scheme without updates [59], we can build a candidate construction of multi-evaluation URE as follows:

- We set the initial database D_0 in garbled RAM to the initial circuit and input pair (C_0, x_0) in the URE scheme. The initial updatable randomized encoding of (C_0, x_0) is an encoding of D_0, computed under garbled RAM scheme, along with an encoding of (C_0, x_0) computed under the standard randomized encoding scheme.
- In order to compute an encoding $\langle \mathbf{u}_i \rangle$ for an update $\mathbf{u}_i$, we compute an encoding $\widetilde{M}_i$ of a machine M_i w.r.t. the garbled RAM scheme where the machine M_i has $\mathbf{u}_i$ hardcoded in it. The machine M_i on input $D_{i-1} = (C_{i-1}, x_{i-1})$ first updates the database to $D_i = (C_i, x_i)$, where $(C_i, x_i) \leftarrow \mathsf{Update}(C_{i-1}, x_{i-1}; \mathbf{u}_i)$, and outputs a fresh standard randomized encoding of (C_i, x_i).

Let us inspect the above solution closely; specifically, the complexity of the machine M_i corresponding to an update $\mathbf{u}_i$. Since M_i computes a fresh (standard) randomized encoding "on-the-fly," in order to achieve the necessary efficiency guarantee for URE, we will require that the encoding time for M_i is independent of its running time. Such a garbled RAM scheme is called a *succinct* garbled RAM scheme [13, 25]. Furthermore, since the output of M_i consists of a fresh randomized encoding, we will also require that the time of encode M_i is independent of its output length. Such a garbled RAM scheme is referred to as *output-compressing* [3, 50].

Recent works [3, 50] show that output-compressing succinct garbled RAM (with sub-exponential security) imply indistinguishability obfuscation (iO). Furthermore, the only known constructions for such a garbled RAM scheme are based on iO, which, in turn seems to require sub-exponential hardness assumptions. Our goal, however, is to obtain a solution for URE using *polynomial hardness assumptions*. As such, the above is not a viable solution for us.

Garbled RAM meets Delegation of Computation. Towards that end, our next idea is to instantiate the above approach using a non-succinct garbled RAM

scheme where the size of the encoding of a machine M_i depends on the running time and the output length of M_i. Such garbled RAM schemes are known to exist based on only one-way functions. At first, it is not clear how to make this approach work since the efficiency requirements of URE are immediately violated.

Towards that end, our next idea is to delegate the computation of the encoding of M_i to the receiver. We implement this idea by using secret-key functional encryption [55, 19, 54]. Roughly speaking, the initial encoding of $C_0(x_0)$ now corresponds to a database encoding of $D_0 = (C_0, x_0)$ w.r.t. a non-succinct garbled RAM scheme along with FE functional key for a circuit P that takes as input an update string $\mathbf{u}_i$ and outputs an encoding $\widetilde{M}_i$ of the machine M_i (as described before). Encoding of an update $\mathbf{u}_i$ now corresponds to an FE encryption of $\mathbf{u}_i$.

In order to achieve the necessary efficiency guarantee of URE, we require that the secret-key FE scheme used above is *compact*, i.e., where the running time of the encryption algorithm on a message m is a fixed polynomial in the length of m and the security parameter, and in particular, independent of the size complexity of any function f in the function family supported by the FE scheme. Indeed, if this were not the case, then the encoding time for an update $\mathbf{u}_i$ in the above solution would depend on the size of the circuit C, which in turn depends on the running time and output length of M_i. Therefore, if the FE scheme were not compact, then the efficiency requirements of URE would once again be violated.

As discussed earlier, a secret-key compact FE scheme with polynomial hardness can be built from polynomial hardness assumptions on multilinear maps using using the results of [33] and [15, 4].

Challenges in Proving Security. While the above construction seems to achieve correctness, it is not immediately clear how to argue security. Note that the circuit P computed by an FE key in the above construction contains the garbling key of the garbled RAM scheme hardwired inside it. Indeed, this is necessary for it to compute the encodings corresponding to machines M_i as discussed above. In order to leverage security of garbled RAM, one approach is to remove the garbling key from the FE function key. However, in order to maintain functionality, this would require hardwiring the output of P, either in the FE key, or in the FE ciphertext. We cannot afford to hardwire the output in the ciphertext since that would violate the efficiency requirements of URE. Thus, our only option is to hardwire the output in the FE key. Note, however, that in the setting of multiple updates, we have to deal with multiple outputs. In particular, the above approach would require hardwiring all the outputs (one corresponding to each update) in the FE key. Doing so "at once" would require putting a bound on the number of updates.

A better option is to hardwire the outputs "one-at-a-time," analogous to many proofs in the iO literature (see, e.g., [39, 3, 23]). Implementing this idea, however, would require puncturing the garbling key. Such a notion of key puncturing is not supported by standard garbled RAM schemes.

Using Cascaded Garbled Circuits. Towards that end, we take a step back and revisit our requirements from the garbled RAM scheme. Our first observation is that in the above solution template, machine M_i need not be a RAM since we are already requiring it to read the entire database! Instead, the key property of garbled RAM with persistent memory that is used in the above template is its ability to maintain *updated state* in the form of encoded database.

We now discuss how to implement this property in a more direct manner by "downgrading" the garbled RAM to a cascaded garbled circuit. Along the way, we will also address the security issues discussed above. Very briefly, we modify the above construction as follows: consider a circuit Q_i that has an update string $\mathbf{u}_i$ hardwired in its description. It takes as input (C_{i-1}, x_{i-1}) and outputs two values. The first value is a fresh randomized encoding of $C_i(x_i)$ where $(C_i, x_i) \leftarrow \mathsf{Update}(C_{i-1}, x_{i-1}; \mathbf{u}_i)$, and the second value is a set of wire keys for the string (C_i, x_i) corresponding to a garbling of the circuit Q_{i+1} (that is defined analogously to Q_i). The initial encoding of $C_0(x_0)$ now corresponds to the input wire keys for the string (C_0, x_0) corresponding to a garbling of circuit Q_1 as defined above, as well as an FE key for a function f that takes as input $\mathbf{u}_i$ and outputs a garbling a circuit Q_i. The encoding of an update $\mathbf{u}_i$ now corresponds to an FE encryption of $\mathbf{u}_i$ as before.

We prove the security of the above construction with respect to indistinguishability based security definition. Simulation-based security can be argued via a generic transformation following [26]. Let C_0^0, C_0^1, x be the initial circuits and input submitted by the adversary in the security proof. And let, $(\mathbf{u}_1^0, \mathbf{u}_1^1), \ldots, (\mathbf{u}_q^0, \mathbf{u}_q^1)$ be the tuple of updates. There are two "chains" of updating processes with the 0^{th} chain starting from C_0^0 and 1^{st} chain starting from C_1^1. The i^{th} "bead" on 0^{th} (resp., 1^{st}) chain corresponds to update $\mathbf{u}_i^0$ (resp., $\mathbf{u}_i^1$).

In the security proof, we start with the real experiment where challenge bit 0 is used. That is, the 0^{th} chain is active in the experiment. In the next step, we introduce the 1^{st} chain, along with the already present 0^{th} chain, into the experiment. However even in this step, 0^{th} chain is still active – that is, generating the randomized encoding at every step is performed using the 0^{th} chain. In the next intermediate hybrids, we slowly switch from 0^{th} chain being activated to 1^{st} chain being activated. In the i^{th} intermediate step, the first i beads on 1^{st} chain are active and on the 0^{th} chain, all except the first i beads are active – this means that the first i updated randomized encodings are computed using the 1^{st} chain and the rest of them are computed using 0^{th} chain. At the end of these intermediate hybrids, we have the 1^{st} chain to be active and 0^{th} chain to be completely inactive. At this stage, we can remove the 0^{th} chain and this completes the proof.

The two chains described above are implemented in a sequence of garbled circuits, that we call *cascaded* garbled circuits. That is, every i^{th} garbled circuit in this sequence produces wire keys for the next garbled circuit. Every garbled circuit in this sequence is a result of $\mathsf{ApplyUpd}$ procedure and encapsulates, for some i, the i^{th} beads on both the chains. In order to move from the i^{th} interme-

diate step to $(i+1)^{th}$ intermediate step, we use the security of garbled circuits. But since these garbled circuits are not given directly, but instead produced by a FE key, we need to make use of security of FE to make this switch work.

3 Preliminaries

We denote the security parameter by λ. The background for randomized encodings and private key (function hiding) functional encryption can be found in the full version [2].

3.1 Updatable Circuits

A boolean circuit C is an directed acyclic graph of in-degree at most 2 with the non-leaf nodes representing $\vee$ (OR), $\wedge$ (AND) and $\neg$ (NOT) gates and the leaf nodes representing the input variables and constants 0 and 1. The nodes with no outgoing edges are designated to be output gates. The size of a circuit $|C|$ is the number of nodes in the graph. Each node is labeled with a different index between 1 and $|C|$. The evaluation of C on input x is performed by first substituting the leaf nodes with the value x and then evaluating gate-by-gate till we reach the output gates. The joint value of all the output gates determine the output of the circuit. Circuit C is said to represent a function $f : \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$ if for every $x \in \{0,1\}^\lambda$ we have $C(x) = f(x)$. We assume that the class of all boolean circuits for every fixed size $|C|$ and n inputs has an efficient binary representation $\mathsf{binary}(C) \in \{0,1\}^{O(|C|)}$. That is, there is an efficient algorithm that computes $C \mapsto (n, |C|, \mathsf{binary}(C))$, and its inverse.

We define the notion of updatable circuits next. A family of updatable circuits $\mathcal{C}$ has associated with it a class of updates $\mathcal{U}$. Given any circuit $C \in \mathcal{C}$ we can transform this circuit into another circuit $C' \in \mathcal{C}$ with the help of an update $\mathbf{u} \in \mathcal{U}$. The updating process could, for instance, change one of the output gates from $\vee$ to $\neg$, change all the gates to $\wedge$ gates and so on. Formally,

Definition 1 (Updatable Circuits). *Consider a circuit family $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$, where $\mathcal{C}_\lambda$ contains $\mathrm{poly}(\lambda)$-sized boolean circuits $C : \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$. Consider a set system of strings $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$, where $\mathcal{U}_\lambda$ is a set of strings of length $\mathrm{poly}(\lambda)$. We define $\mathcal{C}$ to be $(\mathsf{Upd},\mathcal{U})$-**updatable** if $C' \leftarrow \mathsf{Upd}(C, \mathbf{u} \in \mathcal{U}_\lambda)$ is also a boolean circuit with input domain $\{0,1\}^\lambda$ and output domain $\{0,1\}^{\ell(\lambda)}$.*

The size of update $\mathbf{u}$ could potentially be much smaller than the size of the circuit C. For instance, the length of the instruction to change all the gates in C to $\wedge$ gate is in fact independent of $|C|$.

4 Updatable Randomized Encodings

We define the notion of updatable randomized encodings (URE) next. Since this notion deals with transforming circuits, this notion will be associated to a class of updatable circuits. But to also capture the joint updatability of both the circuit and the input together, we introduce the notion of hardwired circuits below.

Hardwired Circuits. A hardwired circuit, associated to a circuit C and input x, takes no input but upon evaluation yields a fixed output $C(x)$.

We provide the formal definition of hardwired circuits below.

Definition 2 (Hardwired Circuit). *Consider a circuit* $C : \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$ *and* $x \in \{0,1\}^\lambda$*. We define a* **hardwired circuit***, denoted by* $C[x]$*, to be a circuit such that,*

- *it takes no input.*
- *upon evaluation (always) outputs* $C(x)$*.*

We interchangeably use $C[x]$ *to denote the circuit as well as the output* $C(x)$ *it computes.*

Two hardwired circuits $C_0[x_0]$ and $C_1[x_1]$ are *equivalent* if and only if $C_0(x_0) = C_1(x_1)$ and $|C_0| = |C_1|$. If $C_0[x_0]$ and $C_1[x_1]$ are equivalent then they are denoted by $C_0[x_0] \equiv C_1[x_1]$. We can generalize this notion and define a class of hardwired circuits as stated below.

Definition 3. *Consider a circuit family* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$*. We define a* **hardwired circuit family** $\{\mathcal{C}[X]_\lambda\}_{\lambda\in\mathbb{N}}$ *where* $\mathcal{C}[X]_\lambda$ *comprises of hardwired circuits of fixed input length and is associated with a bijective function* $\phi : \mathcal{C}_\lambda \times \{0,1\}^\lambda \to \mathcal{C}[X]_\lambda$ *such that if* $\phi\left(C \in \mathcal{C}_\lambda, x\right) = \mathbf{C}$ *then the output of the hardwired circuit* $\mathbf{C}$ *is* $C(x)$*.*

We can now talk about updatability of hardwired circuits. Note that this captures joint updating of both the circuit as well as the input hardwired into it.

Definition 4 (Updatable Hardwired Circuits). *Consider a family of hardwired circuits* $\{\mathcal{C}[X]_\lambda\}_{\lambda\in\mathbb{N}}$*, where* $\mathcal{C}[X]_\lambda$ *contains* $\mathrm{poly}(\lambda)$*-sized boolean circuits* $C[X] : \perp \to \{0,1\}^{\ell(\lambda)}$*. Consider a set system of strings* $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$*, where* $\mathcal{U}_\lambda$ *contains a set of strings of length* $\mathrm{poly}(\lambda)$*. We define* $\mathcal{C}[X]$ *to be* $(\mathsf{Upd},\mathcal{U})$**-updatable** *if* $\mathbf{C} \leftarrow \mathsf{Upd}\left(C[x], \mathbf{u}\right)$*, where* $C[x] \in \mathcal{C}[X]_\lambda, \mathbf{u} \in \mathcal{U}_\lambda$*, then* $\mathbf{C}$ *is also a hardwired circuit.*

We now proceed to give a formal definition of URE.

Syntax. A scheme $\mathsf{URE} = (\mathsf{Encode}, \mathsf{GenUpd}, \mathsf{ApplyUpd}, \mathsf{Decode})$ for a $(\mathsf{Upd},\mathcal{U})$-updatable class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ is defined below. We denote $\mathcal{C}[X]$ to be the corresponding updatable *hardwired* circuit family.

- **Encode,** $(\langle C[x]\rangle_{\mathsf{ure}}, \mathsf{st}) \leftarrow \mathsf{Encode}\left(1^\lambda, C, x\right)$: On input security parameter λ, circuit $C \in \mathcal{C}_\lambda$, input $x \in \{0,1\}^\lambda$, it outputs the joint encoding $\langle C[x]\rangle_{\mathsf{ure}}$ and state st.

- **Generating Secure Update,** $(\langle \mathbf{u}\rangle_{\mathsf{ure}}, \mathsf{st}') \leftarrow \mathsf{GenUpd}\left(\mathsf{st}, \mathbf{u}\right)$: On input state st, update $\mathbf{u} \in \mathcal{U}_\lambda$, output the secure update $\langle \mathbf{u}\rangle_{\mathsf{ure}}$ along with the new state st'.

– **Apply Secure Update,** $\langle C'[x'] \rangle_{\mathsf{ure}} \leftarrow \mathsf{ApplyUpd}\left(\langle C[x] \rangle_{\mathsf{ure}}, \langle \mathbf{u} \rangle_{\mathsf{ure}}\right)$: On input randomized encoding $\langle C[x] \rangle_{\mathsf{ure}}$, secure update $\langle \mathbf{u} \rangle_{\mathsf{ure}}$, output the updated randomized encoding $\langle C'[x'] \rangle_{\mathsf{ure}}$.

– **Evaluation,** $\alpha \leftarrow \mathsf{Decode}\left(\langle C[x] \rangle_{\mathsf{ure}}\right)$: On input randomized encoding $\langle C[x] \rangle_{\mathsf{ure}}$, output the decoded value α.

We associate the above scheme with efficiency, correctness and security properties. We first talk about the efficiency requirement. Modeling of correctness and security properties is tricky and we will deal with them in a separate subsection.

Efficiency. We lay out different efficiency properties associated with the above scheme.

– *Encoding Time*: This property requires that the encoding time of (C, x) is significantly "simpler" than computing $C(x)$. The efficiency aspect can be quantified in many ways – in this work, we define encoding to be efficient if the depth of Encode circuit is smaller than C.

– *Secure Update Generation Time*: This property requires that the runtime of $\mathsf{GenUpd}\left(\mathsf{st}, \mathbf{u}\right)$ is $p(\lambda, |\mathbf{u}|)$, where p is an a priori fixed polynomial. In other words, the update generation time is independent of the size of the encoded circuit.

– *State Size*: This property requires that the size of the state maintained by the authority is a fixed polynomial in the security parameter. That is, the size of st output by Encode and GenUpd is always $\mathrm{poly}(\lambda)$ independent of the size of the machines and the update sizes.

– *Secure Update Size*: This property states that the size of the secure version of the update should solely depend on the size of the update. Formally, we have the size of the secure update to be $|\langle \mathbf{u} \rangle_{\mathsf{ure}}| = p(\lambda, |\mathbf{u}|)$, where $(\langle \mathbf{u} \rangle_{\mathsf{ure}}, \mathsf{st}') \leftarrow \mathsf{GenUpd}\left(\mathsf{st}, \mathbf{u}\right)$. Note that any URE scheme that satisfies the above secure update generation time property also satisfies this property.

– *Runtime of Update*: Informally, this property states that the time to update the secure encoding incurs a polynomial overhead in the time to update the plaintext circuit. Formally, the runtime of $\mathsf{ApplyUpd}(\langle C[x] \rangle_{\mathsf{ure}}, \langle \mathbf{u} \rangle_{\mathsf{ure}})$ is $p(\lambda, t, |\mathbf{u}|)$, where t is the time taken to execute $\mathsf{Upd}(C[x], \mathbf{u})$.

Our constructions achieve a restricted version of the above properties. On the positive side, our construction in Section 5 achieves the 'Encoding Time' property and 'Secure Update Generation Time' properties. We use a term to define a URE scheme that satisfies the secure update generation time property – we call it *output compact URE*.

Definition 5 (Output Compact URE). *An URE scheme that is said to be* **output compact** *if it satisfies 'Secure update generation time' property.*

In the case of indistinguishability security, our construction will be output-compact, i.e., the updates will be independent of the output length of the circuit. In the case of simulation-based security, our construction will not achieve output compactness. This is, in fact, inherent and a formal lower bound to this effect can be established along the same lines as in [1, 26]. On the flip side, our construction does not satisfy 'Runtime of Update' property.

In the full version [2], we provide a transformation from any URE scheme that satisfies the 'Secure Update Generation Time' property to one that additionally satisfies the 'State Size' property. This transformation uses non-succinct garbled RAMs, and assumes only one-way functions.

4.1 Sequential Updating

We first consider sequential updating process that will be the main focus of this work. For alternate updating processes, refer to the full version [2]. Sequential Updating process allows for updating a randomized encoding using multiple patches in a sequential manner. That is, given secure updates $\langle \mathbf{u}_1 \rangle_{\mathsf{ure}}, \ldots, \langle \mathbf{u}_\ell \rangle_{\mathsf{ure}}$, we can update a randomized encoding $\langle C[x] \rangle_{\mathsf{ure}}$ by first applying $\langle \mathbf{u}_1 \rangle_{\mathsf{ure}}$ on $\langle C[x] \rangle_{\mathsf{ure}}$ to obtain the updated encoding $\langle C_1[x_1] \rangle_{\mathsf{ure}}$; next we apply $\langle \mathbf{u}_2 \rangle_{\mathsf{ure}}$ on $\langle C_1[x_1] \rangle_{\mathsf{ure}}$ to obtain the updated encoding $\langle C_2[x_2] \rangle_{\mathsf{ure}}$ and so on. After all the updates, we end up with the updated encoding $\langle C_\ell[x_\ell] \rangle_{\mathsf{ure}}$.

Correctness of Sequential Updating. Intuitively, the correctness property states that computing the randomized encoding $\langle C[x] \rangle_{\mathsf{ure}}$, applying the secure updates $\langle \mathbf{u}_1 \rangle_{\mathsf{ure}}, \ldots, \langle \mathbf{u}_\ell \rangle_{\mathsf{ure}}$ sequentially and finally decoding yields the same result as the output of the circuit obtained by updating the hardwired circuit $C[x]$ by applying the updates $\mathbf{u}_1, \ldots, \mathbf{u}_\ell$ sequentially. We give the formal description below.

Consider a circuit $C \in \mathcal{C}_\lambda$, input $x \in \{0,1\}^\lambda$. Consider a vector of updates $\mathbf{U} \in (\mathcal{U}_\lambda)^q$, where $q(\lambda)$ is a polynomial in λ. Consider the following two processes:

Secure updating process:

1. $(\langle C[x] \rangle_{\mathsf{ure}}, \mathsf{st}_0) \leftarrow \mathsf{Encode}\left(1^\lambda, C, x\right)$.
2. For every $i \in [q]$; $(\langle \mathbf{u}_i \rangle_{\mathsf{ure}}, \mathsf{st}_i) \leftarrow \mathsf{GenUpd}\left(\mathsf{st}_{i-1}, \mathbf{u}_i\right)$, where $\mathbf{u}_i$ is the i^{th} entry in $\mathbf{U}$.
3. Let $\langle C_0[x_0] \rangle_{\mathsf{ure}} := \langle C[x] \rangle_{\mathsf{ure}}$. For every $i \in [q]$;

$$\langle C_i[x_i] \rangle_{\mathsf{ure}} \leftarrow \mathsf{ApplyUpd}\left(\langle C_{i-1}[x_{i-1}] \rangle_{\mathsf{ure}}, \langle \mathbf{u}_i \rangle_{\mathsf{ure}}\right).$$

Insecure updating process:

1. Let $(C_0, x_0) := (C, x)$. For every $i \in [q]$, we have $C_i[x_i] \leftarrow \mathsf{Upd}(C_{i-1}[x_{i-1}], \mathbf{u}_i)$. The output of $C_q[x_q]$ is $C_q(x_q)$.

We have,

$$\mathsf{Decode}\Big(\langle C_q[x_q] \rangle_{\mathsf{ure}}\Big) = C_q(x_q)$$

Security of Sequential Updating. We consider two different security notions of sequential updatable RE. First, we consider simulation-based notion and then we consider the weaker indistinguishability-based notion.

Our security notions attempt to capture the intuition that an updateable randomized encoding $\langle C_0[x_0]\rangle_{\mathsf{ure}}$ and a sequence of updates $\langle \mathbf{u}_1\rangle_{\mathsf{ure}}, \ldots, \langle \mathbf{u}_q\rangle_{\mathsf{ure}}$ should reveal only the outputs $C_0(x_0), C_1(x_1), \ldots C_q(x_q)$ where C_i and X_i are defined as in the preceding correctness definition. In addition to hiding the circuits and inputs as in traditional randomized encodings, a URE additionally hides the sequence of updates. Our URE construction satisfies this update-hiding property.

We could instead consider a relaxed notion, in which updates are partially or wholly revealed (modifying the definitions appropriately). Indeed, this is what we will do in the context of updatable garbled circuits. In the full version [2], we provide a generic transformation from an update-revealing URE scheme to an update-hiding URE scheme, assuming only the existence of one-way functions.

Simulation-Based Security. We adopt the *real world/ ideal world* paradigm in formalizing the simulation-based security definition of sequential updatable RE. In the real world, the adversary receives a randomized encoding and encodings of updates. All the encodings are generated honestly as per the description of the scheme. In the ideal world, the adversary is provided simulated randomized encodings and encodings of updates. These simulated encodings are generated as a function of the outputs and in particular, the simulation process does not receive as input the circuit, input or the plaintext updates. A sequential updatable RE scheme is secure if an efficient adversary cannot tell apart real world from the ideal world.

The ideal world is formalized by considering a simulator Sim that runs in probabilistic polynomial time. Sim gets as input the output of circuit $C(x)$, the length of C and produces a simulated randomized encoding. We emphasize that Sim does not receive as input C or x. After this, Sim simulates the update encodings. On input length of update $\mathbf{u}_i$, value $C_i(x_i)$, it generates a simulated encoding of $\mathbf{u}_i$. Here, $C_i(x_i)$ is obtained by first updating $C_{i-1}[x_{i-1}]$ using $\mathbf{u}_i$ to obtain $C_i[x_i]$, whose output is $C_i(x_i)$ and also, $C_0[x_0]$ is initialized with $C[x]$. For this discussion, we consider the scenario where the circuit, input along with the updates are fixed at the beginning of the experiment. This is termed as the *selective* setting. We describe the formal experiment in Figure 1.

We present the formal security definition below.

Definition 6 (SIM-secure Sequential URE). *A sequential URE scheme* URE *for* $(\mathsf{Upd},\mathcal{U})$*-updatable class of circuits* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *is said to be* **SIM-secure** *if for every PPT adversary* $\mathcal{A}$*, for every circuit* $C \in \mathcal{C}_\lambda$*, updates* $\mathbf{u}_1, \ldots, \mathbf{u}_q \in \mathcal{U}_\lambda$*, there exists a PPT simulator* Sim *such that the following holds for sufficiently large* $\lambda \in \mathbb{N}$*,*

$$\left|\mathsf{Pr}\left[0 \leftarrow \mathsf{IdealExpt}^{\mathcal{A}}\left(1^\lambda, C, x, \{\mathbf{u}_i\}_{i\in[q]}\right)\right] - \mathsf{Pr}\left[0 \leftarrow \mathsf{RealExpt}^{\mathcal{A}}\left(1^\lambda, C, x, \{\mathbf{u}_i\}_{i\in[q]}\right)\right]\right| \leq \mathsf{negl}(\lambda),$$

where negl *is a negligible function.*

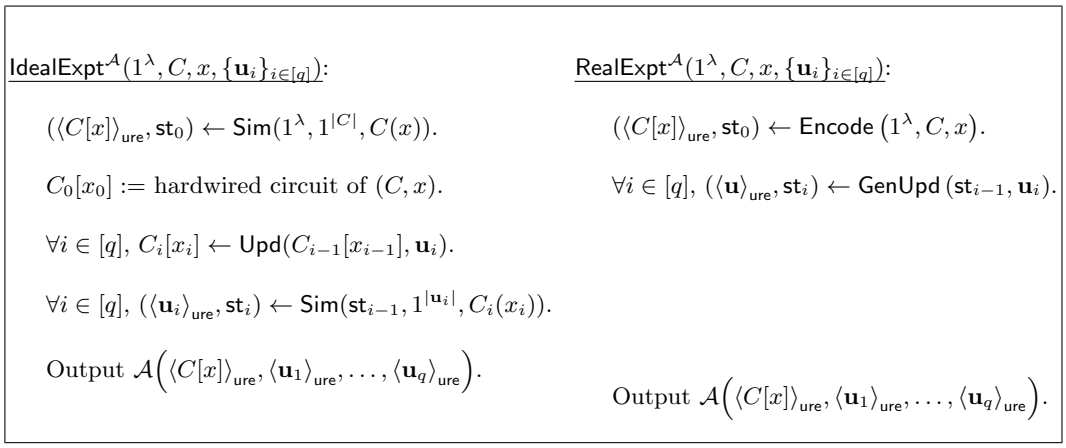

Fig. 1. Selective Simulation-Based Definition of Sequential URE.

We also define indistinguishability-based security notion. We show a transformation from indistinguishability-based security notion to simulation based security notion. This can be found in the full version.

5 Output-Compact URE from FE

In this section, we present our construction of updatable randomized encodings satisfying output compactness properties.

5.1 Construction

Our goal is to construct an updatable randomized encoding scheme, $\mathsf{URE} = (\mathsf{Encode}, \mathsf{GenUpd}, \mathsf{ApplyUpd}, \mathsf{Decode})$ for $\mathcal{C}$. The main tools we use in our construction are the following. We refer the reader to the preliminaries for the definitions of these primitives.

- Randomized Encoding scheme, $\mathsf{RE} = (\mathsf{RE.Enc}, \mathsf{RE.Dec})$ for the same class of circuits $\mathcal{C}$.
- Compact, Function-private, Single-Key, Secret-key functional encryption (FE) scheme,
  $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$.
- Garbling Scheme for circuits, $\mathsf{GC} = (\mathsf{GrbCkt}, \mathsf{GrbInp}, \mathsf{EvalGC})$.

Remark 1. In the full version [2], we show that compact secret-key functional encryption is necessary for our construction of updatable randomized encodings if we believe that learning with errors assumption holds true.

We assume, without loss of generality, that all randomized algorithms require only λ-many random bits. We use the above tools to design the algorithms of URE as given below.

The updatable randomized encoding of (C, x) will consist of a (standard) randomized encoding (C, x) and some additional information necessary to carry out the updating process. This additional information consists of a garbled input encoding of C and x with respect to GC, and a FE secret key for a function that takes as input an update and outputs a garbled circuit mapping C and x to a new randomized encoding and new garbled circuit input encodings of C' and x', which are the updated values. Henceforth, we denote by s the size of the representation of the harwired circuit $C[x]$.

$\underline{\mathsf{Encode}\left(1^\lambda, C, x\right)}$: On input security parameter λ, perform the following operations.

1. Execute the setup of FE, $\mathsf{FE.MSK} \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
2. Compute a functional key $\mathsf{FE.SK}_{\mathsf{RRGarbler}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.MSK}, \mathsf{RRGarbler})$, where $\mathsf{RRGarbler}$ is as defined in Figure 3.
3. In the next step, generate a randomized encoding of input (C, x). That is, compute $\mathsf{RE.Enc}(1^\lambda, C, x)$ to obtain $\langle C[x] \rangle_{\mathsf{re}}$.
4. As stated earlier, let s be the size of the representation of $C[x]$. Generate a garbled circuit input encoding of $(C[x], \perp)$ by evaluating $\langle C[x], \perp \rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbInp}(C[x], \perp; r_{\mathsf{gc}})$, where r_{gc} is the randomness used to garble the input. Here we view $(C[x], \perp)$ as an *input* (to the circuit $\mathsf{RelockRelease}$).
5. Output as the randomized encoding the tuple,

$$\langle C[x] \rangle_{\mathsf{ure}} = \Big(\mathsf{FE.SK}_{\mathsf{RRGarbler}},\ \langle C[x] \rangle_{\mathsf{re}},\ \langle C[x], \perp \rangle_{\mathsf{gc}} \Big)$$

   and set the state to be $\mathsf{st} = (\mathsf{FE.MSK},\ r_{\mathsf{gc}})$.

$\underline{\mathsf{GenUpd}\left(\mathsf{st}_i,\ \mathbf{u}_{i+1}\right)}$: It takes as input the state $\mathsf{st}_i = (\mathsf{FE.MSK}, r_{\mathsf{gc},i})$ and update $\mathbf{u}_{i+1}$.

1. Sample random coins $r_{\mathsf{re},i+1}$ and $r_{\mathsf{gc},i+1}$. Let $\mathsf{mode} = 0$.
2. Generate the FE ciphertext,

$$\mathsf{CT}_{i+1} \leftarrow \mathsf{FE.Enc}\left(\mathsf{FE.MSK},\ (\mathbf{u}_{i+1},\ \perp,\ r_{\mathsf{gc},i},\ r_{\mathsf{gc},i+1},\ r_{\mathsf{re},i+1},\ \mathsf{mode})\right)$$

3. Set the new state $\mathsf{st}_{i+1} = (\mathsf{FE.MSK},\ r_{\mathsf{gc},i+1})$.
4. Output $\langle \mathbf{u}_{i+1} \rangle_{\mathsf{ure}} = \mathsf{CT}_{i+1}$.

$\underline{\mathsf{ApplyUpd}\left(\langle C_i[x_i] \rangle_{\mathsf{ure}}, \langle \mathbf{u}_{i+1} \rangle_{\mathsf{ure}}\right)}$: On input circuit encoding $\langle C_i[x_i] \rangle_{\mathsf{ure}}$ and update encoding $\langle \mathbf{u}_{i+1} \rangle_{\mathsf{ure}} = \mathsf{CT}_{i+1}$, execute the following.

$\underline{\mathsf{RelockRelease}_{i+1}}$

Input: $C_i^0[x_i^0]$, $C_i^1[x_i^1]$
Hard-coded values: $\mathbf{u}_{i+1}^0$, $\mathbf{u}_{i+1}^1$, $r_{\mathsf{gc},i+1}$, $r_{\mathsf{re},i+1}$, and mode

- Update both the hardwired circuits $C_i^b[x_i^b]$ using $\mathbf{u}_{i+1}^b$:

$$C_{i+1}^b[x_{i+1}^b] \leftarrow \mathsf{Upd}(C_i^b[x_i^b], \mathbf{u}_{i+1}^b)$$

- Encode the updated hardwired circuit $C_{i+1}^{\mathsf{mode}}[x_{i+1}^{\mathsf{mode}}$:

$$\langle C_{i+1}^{\mathsf{mode}}[x_{i+1}^{\mathsf{mode}}]\rangle_{\mathsf{re}} \leftarrow \mathsf{RE.Enc}\Big(C_{i+1}^{\mathsf{mode}}[x_{i+1}^{\mathsf{mode}}];\ r_{\mathsf{re},i+1}\Big)$$

- Compute the randomized encoding of the input $\big(C_{i+1}^0[x_{i+1}^0],\ C_{i+1}^1[x_{i+1}^1]\big)$:

$$\langle C_{i+1}^0[x_{i+1}^0],\ C_{i+1}^1[x_{i+1}^1]\rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbInp}\Big(\ (C_{i+1}^0[x_{i+1}^0], C_{i+1}^1[x_{i+1}^1])\ ;\ r_{\mathsf{gc},i+1}\Big)$$

- Output $\Big(\langle C_{i+1}^{\mathsf{mode}}[x_{i+1}^{\mathsf{mode}}]\rangle_{\mathsf{re}},\ \langle C_{i+1}^0[x_{i+1}^0], C_{i+1}^1[x_{i+1}^1]\rangle_{\mathsf{gc}}\Big)$

Fig. 2.

$\underline{\mathsf{RRGarbler}}$

Input: $(\mathbf{u}_{i+1}^0, \mathbf{u}_{i+1}^1, r_{\mathsf{gc},i}, r_{\mathsf{gc},i+1}, r_{\mathsf{re},i+1}, \mathsf{mode})$

Compute the garbled circuit encoding of $\mathsf{RelockRelease}_{i+1}$*, which is defined in Figure 2:*

$$\langle \mathsf{RelockRelease}_{i+1}\rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbCkt}\Big(\mathsf{RelockRelease}_{i+1};\ r_{\mathsf{gc},i}\Big)$$

Output $\langle \mathsf{RelockRelease}_{i+i}\rangle_{\mathsf{gc}}$.

Fig. 3.

1. Parse the circuit encoding as:

$$\langle C_i[x_i]\rangle_{\mathsf{ure}} = \Big(\ \mathsf{FE.SK}_{\mathsf{RRGarbler}},\ \langle C_i[x_i]\rangle_{\mathsf{re}},\ \langle C_i[x_i], \perp\rangle_{\mathsf{gc}}\ \Big)$$

2. Execute the FE decryption, $\mathsf{FE.Dec}(\mathsf{FE.SK}_{\mathsf{RRGarbler}}, \mathsf{CT}_{i+1})$ to obtain:

$$\langle \mathsf{RelockRelease}_{i+1}\rangle_{\mathsf{gc}}.$$

3. Execute the decode algorithm of the garbling scheme,

$$(\langle C_{i+1}[x_{i+1}]\rangle_{\mathsf{re}}, \langle C_{i+1}[x_{i+1}], \perp\rangle_{\mathsf{gc}}) \leftarrow \mathsf{EvalGC}(\langle \mathsf{RelockRelease}_{i+1}\rangle_{\mathsf{gc}}, \langle C_i[x_i]\rangle_{\mathsf{gc}})$$

That is, the decode algorithm outputs the randomized encoding of updated hardwired circuit $C_{i+1}[x_{i+1}]$ and also wire keys of $(C_{i+1}[x_{i+1}], \bot)$ that will be input to the next level garbled circuit.

4. Output $\left(\mathsf{FE.SK}_{\mathsf{RRGarbler}}, \langle C_{i+1}[x_{i+1}]\rangle_{\mathsf{re}}, \langle C_{i+1}[x_{i+1}], \bot\rangle_{\mathsf{gc}}\right)$.

$\underline{\mathsf{Decode}}\,(\langle C_i[x_i]\rangle_{\mathsf{ure}})$: On input encoding

$$\langle C_i[x_i]\rangle_{\mathsf{ure}} = (\mathsf{FE.SK}_{\mathsf{RRGarbler}},\ \langle C_i[x_i]\rangle_{\mathsf{re}},\ \langle C_i[x_i], \bot\rangle_{\mathsf{gc}}),$$

decode the encoding $\langle C_i[x_i]\rangle_{\mathsf{re}}$ by executing $\mathsf{RE.Dec}(\langle C_i[x_i]\rangle_{\mathsf{re}})$ to obtain α. Output the value α.

In the full version [2], we show that the above scheme satisfies all the properties of an updatable randomized encodings scheme.

References

1. Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 500–518, 2013.
2. Prabhanjan Ananth, Aloni Cohen, and Abhishek Jain. Cryptography with updates. Cryptology ePrint Archive, Report 2016/934, 2016. http://eprint.iacr.org/2016/934.
3. Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *Advances in Cryptology–CRYPTO 2015*, pages 308–326. Springer, 2015.
4. Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. Technical report, Cryptology ePrint Archive, Report 2015/730, 2015.
5. Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Patchable indistinguishability obfuscation: io for evolving software. *IACR Cryptology ePrint Archive*, 2015:1084, 2015.
6. Benny APPLEBAUM, Yuval ISHAI, and Eyal KUSHILEVITZ. Cryptogaphy in nc0. *SIAM journal on computing*, 36(4):845–888, 2007.
7. Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609. Acm, 2007.
8. Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2005, San Diego, California, USA*, 2005.
9. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001.

10. Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *Advances in Cryptology—CRYPTO'94*, pages 216–233. Springer, 1994.
11. Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography and application to virus protection. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 45–56. ACM, 1995.
12. Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *Advances in Cryptology—EUROCRYPT'97*, pages 163–192. Springer, 1997.
13. Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Siddartha Telang. Succinct randomized encodings and their applications. In *STOC*, 2015.
14. Nir Bitansky, Ryo Nishimaki, Alain Passelègue, and Daniel Wichs. From cryptomania to obfustopia through secret-key functional encryption. volume TCC, 2016.
15. Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 171–190. IEEE, 2015.
16. Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, pages 127–144, 1998.
17. Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 149–168. Springer, 2011.
18. Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *Advances in Cryptology–CRYPTO 2013*, pages 410–428. Springer, 2013.
19. Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography*, pages 253–273. Springer, 2011.
20. Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography Conference*, pages 535–554. Springer, 2007.
21. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology-ASIACRYPT 2013*, pages 280–300. Springer, 2013.
22. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public-Key Cryptography–PKC 2014*, pages 501–519. Springer, 2014.
23. Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions. In *Theory of Cryptography*, pages 1–30. Springer, 2015.
24. Enrico Buonanno, Jonathan Katz, and Moti Yung. Incremental unforgeable encryption. In *Fast Software Encryption*, pages 109–124. Springer, 2001.
25. Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and RAM programs. In *STOC*, 2015.
26. Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 519–535, 2013.

27. Nishanth Chandran, Bhavana Kanukurthi, and Rafail Ostrovsky. Locally updatable and locally decodable codes. In *Theory of Cryptography*, pages 489–514. Springer, 2014.
28. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In *Advances in Cryptology–EUROCRYPT 2012*, pages 281–300. Springer, 2012.
29. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Succinct malleable nizks and an application to compact shuffles. In *Theory of Cryptography*, pages 100–119. Springer, 2013.
30. Dana Dachman-Soled, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Locally decodable and updatable non-malleable codes and their applications. In *Theory of Cryptography*, pages 427–450. Springer, 2015.
31. Marc Fischlin. Incremental cryptography and memory checkers. In *Advances in Cryptology—EUROCRYPT'97*, pages 393–408. Springer, 1997.
32. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013.
33. Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. *IACR Cryptology ePrint Archive*, 2014:666, 2014.
34. Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 467–476. ACM, 2013.
35. Sanjam Garg and Omkant Pandey. Incremental program obfuscation. *IACR Cryptology ePrint Archive*, 2015:997, 2015.
36. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.
37. Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled ram revisited. In *Advances in Cryptology–EUROCRYPT 2014*, pages 405–422. Springer, 2014.
38. Craig Gentry, Allison Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *International Cryptology Conference*, pages 426–443. Springer, 2014.
39. Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.
40. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.
41. Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 578–602. Springer, 2014.
42. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium*

on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013, pages 555–564. ACM, 2013.

43. Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from lwe. In *Annual Cryptology Conference*, pages 503–523. Springer, 2015.
44. Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 469–477. ACM, 2015.
45. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 89–98, 2006.
46. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 294–304. IEEE, 2000.
47. Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 146–162. Springer, 2008.
48. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 669–684. ACM, 2013.
49. Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation with non-trivial efficiency. In *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*, pages 447–462, 2016.
50. Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. In *Theory of Cryptography*, pages 96–124. Springer, 2016.
51. Steve Lu and Rafail Ostrovsky. How to garble ram programs? In *Advances in Cryptology–EUROCRYPT 2013*, pages 719–734. Springer, 2013.
52. Daniele Micciancio. Oblivious data structures: applications to cryptography. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 456–464. ACM, 1997.
53. Ilya Mironov, Omkant Pandey, Omer Reingold, and Gil Segev. Incremental deterministic public-key encryption. In *EUROCRYPT 2012*.
54. Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.
55. Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 457–473, 2005.
56. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484. ACM, 2014.

57. Elaine Shi, John Bethencourt, TH Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 350–364. IEEE, 2007.
58. Brent Waters. A punctured programming approach to adaptively secure functional encryption. In *CRYPTO 2015*, 2015.
59. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.
60. Mark Zhandry. How to avoid obfuscation using witness prfs. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, pages 421–448, 2016.