Cryptography with Dispensable Backdoors

Kai-Min Chung*[1], Marios Georgiou†[2], Ching-Yi Lai‡[1] and Vassilis Zikas§[3]

[1]Academia Sinica
[2]City University of New York
[3]University of Edinburgh

April 18, 2018

Abstract

Backdooring cryptographic algorithms is an indisputable taboo in the cryptographic literature for a good reason: however noble the intentions, backdoors might fall in the wrong hands, in which case security is completely compromised. Nonetheless, more and more legislative pressure is being produced to enforce the use of such backdoors.

In this work we introduce the concept of *dispensable cryptographic backdoors* which can be used only once and become useless after that. These exotic primitives are impossible in the classical digital world without stateful and secure trusted hardware support, but, as we show, are feasible assuming quantum computation and access to classical stateless hardware tokens.

Concretely, we construct a dispensable (single-use) version of message authentication codes, and use them to derive a black-box construction of stateful hardware tokens in the above setting with quantum computation and classical stateless hardware tokens. This can be viewed as a generic transformation from stateful to stateless tokens and enables, among other things, one-time programs and memories.

We then use the latter primitives to propose a resolution to the most prominent recent legislative push in favor of backdooring cryptography: the conflict between Apple and FBI last year. We show that it is possible for Apple to create a one-time backdoor which unlocks any single device, and no more than one, i.e., the backdoor becomes useless after it is used. We further describe how to use our ideas to derive a version of CCA-secure public key encryption, which is accompanied with a dispensable (i.e, single-use, as in the above scenario) backdoor.

1 Introduction

The use of strong cryptographic primitives for widely available devices has led to controversial debates between the computer security community and public policy makers. On the one hand, law enforcement agencies argue that allowing access to such primitives enables cyber-terrorists to use it to elude detection, and thereby reduces the effectiveness of law enforcement. On the other hand, the computer security—and most vocally the cryptographic—community argues that allowing everyone to use such strong primitives can help protect their security and make cybercrime less effective in the first place. In fact, this debate dates back to the Cold War era and due to its severity has been often referred to as the Crypto Wars [11].

Although the Electronic Frontier Foundation (EFF) legally fought—and won in court—policies which could undermine security of the offered solutions, the debate is holding. And with the wide spread of the Internet, it has been reignited. One of the most high-profile recent cases was that of the FBI against Apple Inc. [18]. In short, the FBI wanted Apple Inc. to create and electronically sign a new software that would enable the FBI to unlock a work-issued iPhone 5C, recovered from one of the shooters in a December 2015 terrorist attack in San Bernardino, CA [21]. Apple Inc. refused to comply with this request. The main

*kmchung@iis.sinica.edu.tw
†mgeorgiou@gradcenter.cuny.edu
‡cylai0616@iis.sinica.edu.tw
§vzikas@inf.ed.ac.uk

argument was that such a software would effectively serve as a backdoor and anyone who got his hands on it would be able to breach the privacy of the smartphone's holder at will. Thus if the backdoor fell in malicious hands, it would yield unprecedented havoc.

The question of "whether or not law-enforcement agencies' selective access—e.g., with a court order—would immediately imply some sort of broken cryptography" has become by its own a matter of debate (see, e.g., [22]). Arguably, unlike what was suggested by the FBI [9], there does not seem to be a way to create some digital information that can only be used by the "good guys" and becomes useless in the hands of malicious actors.

Notwithstanding, in this work we prove that with the help of quantum storage, we can develop backdoors that can be used *only once* to bypass the security of any device from a defined set, and then become completely useless. In particular, we show that these single-use backdoors allow an arbitrary smartphone to be unlocked and they become useless after this phone has been unlocked.

1.1 Our Contributions

We propose a general framework called "dispensable backdoored cryptography," where the goal is to compile standard cryptographic primitives, such as authentication/identification and encryption schemes, into their dispensable backdoored counterparts. At a high-level, for a cryptographic primitive where security is defined in terms of a game between an adversary and a challenger, e.g., CCA secure encryption, we define its "dispensable backdoored" version. In this version the generation algorithm outputs additionally a quantum key that can be used to gain knowledge about the secret information, e.g., the decryption key. Importantly, the security property ensures that in absence of the trapdoor the backdoored scheme achieves the same guarantees as its non-backdoored counterpart. Additionally, anyone given access to the backdoor can use it only once, i.e., an adversary, participating in two CCA games and is given access to the trapdoor might only win in one of these games.

The above transformation is feasible by using the so called *one-time programs* [15]. Informally, these are programs that can be executed once and then become useless, i.e., they terminate and cannot be reused. This exotic primitive is known to be impossible both in the classical world [15] and in the quantum world [7] without any further setup assumptions. We overcome this limitation by defining one-time programs relative to a stateless *classical* oracle—this corresponds to equipping the programs with access to a classical honestly generated stateless hardware token. We prove that one-time programs are possible in this stateless (classical) hardware token model. In fact, our construction of one-time programs from stateless (classical) tokens is a special instantiation of a more general transformation that with the use of quantum computation reduces a stateful (classical) oracle to a stateless (also classical) one.

Our work continues the line of research of investigating the power of stateless tokens which are known to be much weaker than their stateless counterparts. For example, stateful tokens are in general susceptible to resetting attacks. Using the stateful-to-stateless transformation we can prevent such an attack in the quantum world.

Equipped with this, we demonstrate construction of "dispensable backdoored devices" as a way to resolve the smartphone vs. law enforcement conundrum. Concretely, our construction allows a device (e.g., smartphone) vendor to embed in its devices a content locking mechanism, and create (and locally store) a dispensable unlocking backdoor that can unlock exactly one smartphone—any one from a specified set. To make our scheme most general we look at the question of how we can extend the set of devices/smartphones that can be unlocked. An obvious solution would be to create a new one-time backdoor and update every phone in the set we want it to be able to unlock. However, this solution is clearly not scalable, as it requires such an update every time a new (batch of) phone needs to be added to this set. Instead, we provide a mechanism for extending the set of devices that can be unlocked with the existing single-use backdoors without interacting with the ones already in the set.

Finally, we show how to construct dispensable backdoored CCA secure encryption as discussed above to demonstrate the generality of our framework. We believe that cryptography with dispensable backdoors can also be extended to most if not all cryptographic primitives with game-based security definition. We believe that such an extension and investigation of further applications/implications is a interesting future direction.

1.2 Overview of our techniques

The main cryptographic tool for our constructions is *dispensable message authentication codes (DMAC).* This primitive allows someone having a quantum key to use it *exactly once* in order to compute an authentication tag to any message. Although we are the first to define this primitive, it turns out that we can directly use the construction of quantum money scheme by Aaronson and Christiano [2] to implement it and its security directly follows by the work of David and Sattath [10].

We use DMACS to devise a generic reduction of stateful to stateless *classical* oracles. These oracles correspond to the notion of stateless honestly generate (classical) hardware tokens. Concretely, for any given stateful token T we show how to generate code for a stateless token along with one-time (quantum) backdoors, so that we can use them to devise a protocol that implements the stateful token T with unconditional security. We note in passing that in the purely classical setting, stateful tokens are strictly more powerful than stateless. E.g., one-time programs are trivial in the stateful token model—simply run the program inside a token that is instructed to halt after the first use—whereas they can be easily shown impossible from stateless tokens without quantum computation—the standard argument that "if I can run a classical stateless program on any one input I can also run it on any two inputs" [15] trivially applies here. Importantly, the tokens/oracles considered in this work can only be queried in a *classical* manner. In fact, such a reduction would become impossible if quantum superposition queries to the stateless token are allowed (see further discussion below), and restriction to classical queries is the key to enable our transformation. We believe that such restriction is a mild assumption that holds in most existing instantiations of such primitives such as smartcards, trusted co-processors etc.

The idea behind our stateful-to-stateless transformation is to generate a (quantum) DMAC tagging key to be used to emulate each round of interaction with the stateful token/oracle T. Denote this key sequence by $k_1, \ldots, k_m$. We then derive (a program for) our stateless oracle which given any sequence of inputs $(x_1, t_1), \ldots, (x_q, t_q)$, checks that each t_i is a valid DMAC tag corresponding to the i-th instance, and if this is the case, performs the same computation that T would on input $x_1, \ldots, x_q$. The security of the DMAC will ensure that none can receive responses from the stateless tokens on query-sequences that have different prefixes. Hence, once any sequence of q queries has been successfully submitted, there is no way to "rewind" the token and query it on a different sequence, which emulates the behavior of the stateful token T.

Having built such a stateful-to-stateless transformation, we go on to create a transformation from any program to its one-time version relative to a classical oracle. The idea here is to first describe the one-time program as a stateful oracle/token and then use the above transformation to turn it into a stateless one. Finally, using one-time programs and one-time memories as their special case, we are ready to attack our original problem of building one-time backdoors for unlocking phones. To achieve this, we allow the vendor to create phones that are hardcoded with a fresh symmetric-encryption key. When a phone locks, it uses this key to encrypt its state and subsequently discards the key. Now, the key can only be accessed through the one-time memory.

In order to allow extendability of our construction, i.e. the vendor can create new phones at any time, we define a notion of extendability of one-time memories. Informally, an extendable OTM is an OTM that can be updated so that it encodes one more secret. We achieve this by having n (1-out-of-2) OTM for n secrets. Each (1-out-of-2) OTM i encodes two things. First, a random key k_i. Second, the actual secret s_i xor'ed with all previous keys $k_1 \oplus \ldots \oplus k_{i-1}$. Informally, in order to extract the secret s_i, we first need to extract all keys k_j for $j < i$, thus destroying all previous OTM.

Finally, we define the notion of encryption with dispensable backdoors and we show how to achieve this primitive out of IND-CCA encryption and one-time programs. The security definition now asks from an adversary to break two challenge ciphertexts given only one backdoor. We finally construct such a scheme by creating a one-time program for the decryption algorithm and prove formally that this construction is secure.

1.3 Related Literature

Our work combines elements from several different areas of classical and quantum cryptography, ranging from quantum money to quantum tokens for message authentication codes and one time memories.

An important element in our construction is what we call quantum dispensable message authentication codes (DMAC). Those can easily be thought as the symmetric key version of the one-time tokens for digital signatures which was proposed by Ben-David and Sattath [10]. It can also be viewed as a generalization of secret-key quantum money in the following sense: in quantum money, an adversary given a quantum state corresponding to a coin should not be able to come up with another valid coin. In quantum tokens for MACs, an adversary should not be able to come up with two tags (for two different messages) given only one token. Clearly, impossibility of tagging two messages implies impossibility of copying the quantum state. Importantly, the adversary should be also given oracle access to the verification algorithm. Related works that implicitly contained this relation is that of Gavinsky [12] and that of Pastawski [20]. Gavinsky calls this primitive a quantum retrieval game (QRG) and it differs from a DMAC in the sense that it does not allow the adversary to have access to a verification oracle. Gavinsky proved that such a primitive is enough to construct secret-key quantum money by having a 3-round verification protocol with the bank. Later Georgiou and Kerenidis [13] improved this construction by achieving only one round verification.

Gordwasser et al. [15] introduced the notion of one-time memories (in short, OTMs); these are devices that contain two secrets but only one of the two can be extracted. They proved that OTMs are enough to achieve one-time programs, i.e. programs that can be run only once. Broadbent et al. [7] extended this result to the quantum setting by showing that quantum OTMs are enough to construct quantum one-time programs.

Interestingly, despite being very close to a non-interactive version of oblivious transfer, OTMs are impossible to achieve in the plain model, both in the classical world and in the quantum world, even in the computational setting. The classical impossibility comes from the fact that a memory should correspond to a classical bitstring and therefore by copying the bitstring we can easily extract both secrets.

Quantumly, the no-cloning theorem could potentially give a way to construct such a primitive. Unfortunately, this is still impossible, since it is theoretically possible to extract a value from a quantum state with probability close to 1 without collapsing the state, and thus we can invert the extraction procedure and then extract the second value as well [1]. Since this impossibility is the motivation for the necessity of classical tokens to achieve one-time primitives (i.e., programs and memories), we sketch the proof intuition in the following (For more details see Aaronson's "Almost as good as new lemma" [1].): Measuring a quantum state returns a random outcome with probability that is proportional to the inner product of the state and the corresponding measurement operator. After the measurement the state collapses to that outcome. The correctness of the OTM, requires that this measurement returns a outcome with high probability. This in turn means that the inner product between the state and the particular measurement operator equals to (say) 1 and thus the measurement does not destroy our state at all. Therefore, an adversary after retrieving the first secret can undo the unitary applied to the quantum state, and then run the second unitary followed by a measurement to retrieve the second secret. Note that this argument shows that stateful-to-stateless transformation is impossible if quantum query to the stateless token (and its inverse) is allowed.

The above quantum-token rewinding attack indicates that shifting from stateless to stateful quantum tokens might not have such a big effect on the feasibility landscape as the switch from stateless to stateful classical tokens. The natural question of what the minimal assumptions are regarding the adversarial model in order to achieve one-time memories—for example, can we achieve OTM relative to a classical oracle that allows only classical queries?— was first address by Liu [19], who proved that OTMs are possible in the isolated-qubits model, where the adversary is restricted to act on each individual qubit of the (quantum) OTM separately.

A more general attempt to address the above problem was taken by Broadbent et al. [6] which initiated the study of feasibility of OTMs in the quantum setting from stateless trusted hardware tokens. Concretely, [6] proposes a definition of one-time memories in the UC framework. The proposed OTM functionality behaves as an oblivious transfer functionality, with the difference that the sender is not notified when the receiver gets one of the messages. Most interestingly, they proposes a construction of OTM and prove that is satisfies their proposed security notion. Although our work is also in the setting of quantum computation with stateless classical token, it differs from [6] in its goals, definitions, and approach it takes as well as in the underlying techniques.

In terms of the goals, our aim is to equip cryptographic primitives with dispensable backdoors. This, in general, requires one time programs (cf. Section 5.2.) Although the latter are known to be reducible to one-time memories, the reduction is achieve by means of a garbled-circuit-based approach. Note that in the garbled-circuit approach, we need an OTM –and thus one token– per gate. Here, we show how to construct one-time programs directly, without going through one-time memories and using a single token, thus our construction is more efficient.

In terms of the used techniques, one of our main contribution is the reduction of stateful to stateless tokens which uses only dispensable MACs (DMACs) which we introduce in this work. As we show, DMACs can be easily constructed and proved secure using techniques from the existing quantum literature, in particular from [10], in a black-box manner. This allows us to avoid the proof complications apparent in [10].

Last but not least, the UC security definition proposed in [6] seems to introduce some inconsistencies in the security statements. Concretely, [6] proves security of their construction only against a corrupted receiver. And it seems to be impossible to argue security for the proposed OTM functionality against a malicious sender. Indeed, to simulate a corrupted sender, the simulator would have to extract the two values from the sender, which appears not to be possible in the construction. Furthermore, one cannot claim that since the token is assumed to be honestly generated the only interesting case is that of an honest sender, as this assumption trivializes the problem. Indeed, with the assumption of an honest sender the functionality can be trivially implemented by assigning the sender the role of a trusted party which does not report to the environment when the receiver receives his secret. We avoid these issues in this work by proposing a property-based definition of one-time memories and programs in the hardware token model, which treats the constructions are oracle algorithms.

Finally, strong, i.e., virtual black-box (VBB) obfuscation can yield a method to go from secret-key to public-key quantum money, can also be used as a way to go from one-time tokens for MACs to one-time tokens for signatures as shown by the work of Ben-David and Sattath [10]. In particular, they showed that it is enough to achieve average case virtual black-box obfuscation for the subspace membership problem. Unfortunately, it is impossible to achieve a general purpose VBB obfuscator [3] and only a handful of functions are known to be are known to be VBB obfuscatable today [23, 8, 24, 17] and none of them seems to be able obfuscate the oracle of [2] that would yield a construction of DMACs.

1.4 The model

We complete this section by describing our model of computation and attack. Before that, we provide some necessary terminology and notation: A function f is negligible if $f(n) \in o(1/\mathsf{poly}(n))$ for any polynomial poly. For two quantum states ρ, σ we denote by $\Delta(\rho, \sigma)$ their trace distance $\frac{1}{2}||\rho - \sigma||_1$. If the trace distance between two quantum states is negligible then we will denote this by $\rho \sim_s \sigma$ and we will say that the quantum states are statistically indistinguishable. For two quantum algorithms A, B that possibly have inputs and oracle access to some algorithms, we will write $A \sim_s B$ if the quantum states that they output are statistically indistinguishable.

Our results consider systems that might be classical and/or quantum computation enabled. E.g., our iQ-phone application considers a classical smartphone but the vendor—who also stores the one-time backdoor—can perform quantum computation, i.e., create, store, and measure qubits. We consider information-theoretic security in the quantum setting, i.e., the adversary is a computationally unbounded quantum machine and the (quantum enabled) parties can transfer quantum states between each other. On the other hand, we assume that the adversary can only make polynomial number of (classical) queries to the stateless token.

Similarly to [6] our constructions are assumed access to *classical* tokens which allow only classical access—in particular, the only way to interact with such a token is to hand it as input a classical string. However, to avoid the aforementioned complications we devise natural definitions of such primitives as oracle algorithms in the plain (non-UC) model of computation. Recall that the assumption of classical-only tokens is not just consistent with the capabilities of common devices that can be used as hardware tokens, e.g., smartcards, but it is also minimal since quantum-accessible tokes are known to be insufficient to circumvent the impossibility of one-time primitives [1].

We model algorithms with access to classical tokens as oracle algorithms, where the oracle is classical and offers the same functionality as the corresponding token. In particular, our classical oracles can only be queried with classical string (no quantum interfaces), may only perform classical computations, and produce classical output. Throughout this work we use the terms oracle and token interchangeably.

2 Dispensable MACs

In this section we introduce the notion of *dispensable message authentication codes* (DMACs, in short) and demonstrate how they can be implemented. In a nutshell, DMACs are a one-time version of classical MACs, i.e., the secret key can be used to authenticate only a single message and then becomes useless (except for verification purposes.) We remark that DMACs are different from what is called one-time MACs in the cryptographic literature. Indeed, the latter are MACs that preserve their security as long as they are used at most once, i.e., they could be used for tagging more than one message but this would render them insecure/forgeable. Instead, DMACs do not allow anyone—honest or adversarial—to use the same MAC key to tag two different messages.[1]

Concretely, classical message authentication codes (MACs) are symmetric-key primitives that allow two parties, who share a key, to exchange messages in an authenticated manner. In a nutshell, any of the parties can use the key within a tagging algorithm Tag to create an authentication tag t to any given message (t is often referred to as a *MAC tag*). The security of the scheme ensures that only the message/tag pairs generated with the shared key will always be accepted by the receiver (completeness); however, no adversary who does not know the key can forge an acceptable authentication tag on a new message (existential unforgeability).

DMACs are MACs whose key can be used to tag exactly one message. This is achieved by making a quantum state as a part of the tag-generation key. This quantum state allows whoever holds it to tag any one message of their choice. We remark that DMACs authenticate classical (not quantum) messages. The formal definition follows.

Definition 1 (Dispensable single-bit MACs). *A single-bit dispensable MAC (DMAC) is a triplet of algorithms* $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ *defined as follows:*

- $\mathsf{Gen}(1^n) \to (s, \rho)$ *is a quantum algorithm that takes as input a security parameter n and returns a dispensable (secret) key-pair consisting of a classical bit-string s of size n and a quantum state ρ. We will refer to ρ as the dispensable (part of the) key.*
- $\mathsf{Tag}(\rho, b) \to t$ *is a quantum algorithm that takes as input a quantum state ρ and a bit b, and returns a classical tag t.*
- $\mathsf{Ver}(s, b, t) \to \{0, 1\}$ *is a classical algorithm that takes as input a secret s, a bit b, and a tag t, and either accepts or rejects.*

The security of DMACs is similar to the security of the original MACs (we refer to [14] for a formal definition), but instead of existential unforgeability, it requires *dispensable existential unforgeability* property which forbids the adversary from creating valid tags for two different (classical) messages with a single tagging key. Note that unlike standard EUCMA-security, the adversary is not given access to a MAC-tag generation oracle—since the dispensable key can be used only once. However, we do allow the adversary to use a classical verification oracle that, given a received (message,tag)-pair (b, t), responds whether or not $\mathsf{Ver}(s, b, t) = 1$.

Definition 2 (Security of DMACs). *A DMAC (scheme)* $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ *is said to be secure if it satisfies the following properties:*

Completeness. *Let (s, ρ) be the output of* Gen*; then for any bit b, it holds that*

$$\mathsf{Ver}(s, b, \mathsf{Tag}(\rho, b)) = 1.$$

Dispensable Existential Unforgeability (DEU). *Let V_s be a classical oracle, which on input a bit b and a tag t, outputs* $\mathsf{Ver}(s, b, t)$. *A DMAC* $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ *is DEU-secure if for any (computationally*

[1] There is an unfortunate clash in terminology in the literature as one-time programs and one-time memories achieve a similar "one-timeness" as DMACs, which is different from what one-time MACs and one-time signatures achieve. Here, we choose to use the term dispensable for MACs to avoid ambiguity.

unbounded) quantum algorithm A with oracle access to V_s and polynomially many queries to V_s, it holds that $\mathsf{Adv}_A^{DMAC} \leq \mathsf{negl}(n)$*, where*

$$\mathsf{Adv}_A^{DMAC} := \Pr_{\substack{(s,\rho)\overset{\$}{\leftarrow}\mathsf{Gen}(1^n)\\(t_0,t_1)\overset{\$}{\leftarrow}A^{V_s(\cdot,\cdot)}(\rho)}}[\mathsf{Ver}(s,0,t_0)=1 \wedge \mathsf{Ver}(s,1,t_1)=1].$$

Note that in the above experiment, the adversary is not given the secret verification key s generated by Gen. This is the reason why this primitive is a secret-key primitive. In fact, in our constructions, if the adversary would get s, then he would be trivially able to generate valid MACs. This is true not only because he is unconditional—he can bruteforce the tag space, but also because our constructions generate the dispensable part of the key ρ from the secret s. It is also easy to verify that the DEU-security implies the classical notion of existential unforgeability [16] but without the MAC-tag generation oracle. Indeed, if the adversary had a process A for generating a valid MAC tag on a message without knowing any part of the key, then he could trivially break DEU-security by first running A to forge a MAC on one message b_0 and then use the dispensable key ρ to generate a MAC tag for b_1 (the completeness property ensures that the latter will always succeed).

Construction of DMACs. Despite our work being the first to provide a formal definition of DMACs, there are a couple of heavily related primitives studied in the quantum cryptography literature—e.g., *quantum retrieval games* [12], *unforgeable quantum tokens* [20], and one-time quantum digital signatures [10]. In fact as part of their one-time quantum digital signatures, David and Sattath [10] already developed the techniques and implicitly defined the algorithms that one needs for implementing DMACs. For completeness, we include this construction and the security argument in Appendix 6. Notice that although we can create directly dispensable MACs from dispensable signatures, such an approach loses the information theoretic security of the definition (since public-key signatures require computational assumptions). Instead, by being careful and using only part of [10], we achieve information theoretic security.

Theorem 1. *There exists a secure single-bit DMAC in the plain model.*

From single-bit to string DMACs. Definition 1 can be extended to the case where we want to tag a string of several (polynomially many) bits. In this case we require that there is no algorithm that can tag two different bit-strings. We refer to this primitive as *DMAC for strings* or *string DMAC*. The corresponding scheme and security definitions are trivially derived by modifying Definitions 1 and 2 so that instead of bits b, b_0, and b_1, they are applied to strings m, m_0, and m_1. For the remainder of this paper, we use DMAC to refer to string DMAC.

The construction of string DMACs from single-bit DMACs is straightforward: To generate tags for an n-bit string $m \in \{0,1\}^n$, simply create n independent key-pairs $(s_1,\rho_1),\ldots,(s_n,\rho_n)$ for single-bit DMACs; the ith dispensable key ρ_i is used to authenticate the i-th bit of m. The security intuition of the construction follows from the fact that since the key-pairs are honestly and independently generated, the single-bit DMAC schemes can be trivially executed in parallel. Note that as straightforward as this might be in the classical setting, quantum interference requires special treatement. Nonetheless, as our DMAC construction is effectively extracted from the one-time quantum signature from [10], the proof follows immediately from their reduction of multi-bit to single-bit one-time quantum signatures [10, Section 5].

Corollary 1. *There exists a secure DMAC for strings in the plain (quantum) model.*

3 Reducing stateful to stateless oracles

Here we show that the notion of quantum DMACs is powerful enough to turn any stateful and classically-queried classical oracle into a stateless one. We model a stateful oracle as a stateless oracle together with a stateful database that stores the queries. Then every time the oracle is queried, it is reset and then runs all the previous queries, followed by the last one.

Using this formalization the transformation of a stateful algorithm A into a stateless B works as follows. As a first step assume some polynomial number q of queries are allowed. We create q single-bit DMAC

```
$A:$
  $i \leftarrow 1$
  $state \leftarrow \perp$
  **loop**: On query $x$
      $(y, state) \leftarrow C_i(x, state)$
      $i++$
      **return** $y$
```

Figure 1: Stateful Algorithm with respect to $\{C_i\}_{i \in \mathbb{Z}}$

```
$A:$
  $S \leftarrow []$
  **loop**: On query $x$
      Append $x$ on $S$ and parse $S$ as $(x_1, \ldots, x_\tau)$
      $state \leftarrow \perp$
      **for** $i \in [\tau]$ **do**
          $(y, state) \leftarrow C_i(x_i, state)$
      **return** $y$
```

Figure 2: Equivalent formulation of stateful Algorithm with respect to $\{C_i\}_{i \in \mathbb{Z}}$

key-pairs (sk_i, ρ_i). Then the algorithm B has the following structure. At the first time it is called, it is queried with x_1 together with a tag t_1 on x_1 with respect to the key sk_1. If the tag is valid, then the algorithm runs as a subroutine A with input x_1 and returns A's output. For the next query x_2, the calling algorithm should provide both (x_1, t_1) and (x_2, t_2), where t_2 is a tag of x_2 with respect to sk_2. Now B will first run A on the first input and then run A on the second input and return this result.

Stateful oracle. A stateful oracle can be thought of as a sequence of stateless oracles $\{C_i\}$, where each of them after execution outputs a state that is fed as input to the next oracle together with a query. Equivalently, a stateful oracle could keep a list of all the previous queries and re-execute the whole computation from the beginning for each new query.

Definition 3 (Stateful algorithm)**.** *A stateful oracle A with respect to a family of stateless oracles $\{C_i\}_{i \in \mathbb{Z}}$ works as follows (fig. 1):*

Up to a polynomial slowdown, an equivalent formulation of a stateful oracle is the following (fig. 2):

Stateful to stateless transformation A stateful to stateless oracle transformation is an algorithm that takes as input the description of a stateful oracle and returns the description of a stateless oracle together with a quantum state. We require the correctness that any algorithm with oracle access to the stateful algorithm can be simulated by another algorithm with oracle access to the stateless one. We also require the security that an algorithm with access to the stateless oracle does not have extra power over one that has access to the stateful one.

Definition 4 (Stateful to Stateless transformation)**.** *Let $\mathcal{A}$ be a family of stateful oracles $\{A_i\}_{i \in \mathbb{Z}}$.* Gen *is a stateful to stateless oracle transformation with respect to $\mathcal{A}$ if there is a family $\mathcal{B} = \{B_j\}_{j \in \mathbb{Z}}$ of stateless oracles such that:*

- $\mathsf{Gen}(1^n, i) \to (\rho, j)$ *is an algorithm that takes as input a security parameter n as well as an index i that corresponds to the stateful oracle A_i and returns a quantum state ρ together with an index j that corresponds to the stateless oracle B_j.*

```
$B_{(s_1,\ldots,s_q,i)}((x_1,t_1),\ldots,(x_\tau,t_\tau))$
    **if** $\mathsf{DMAC.Ver}(s_j,x_j,t_j)=0$ for some $j\in[\tau]$ **then**
        **return** "Invalid tag"
    $state \leftarrow \perp$
    **for** $j\in[\tau]$ **do**
        $(y, state) \leftarrow C_{i,j}(x, state)$
    **return** $y$
```

Figure 3: The class of stateless oracles $\mathcal{B}$

```
$\mathsf{Gen}(1^n, i)$
    **for** $j\in[q]$ **do**
        $(s_j,\rho_j) \leftarrow \mathsf{DMAC.Gen}(1^n)$
    **return** $((s_1,\ldots,s_q,i),\rho_1\otimes\ldots\otimes\rho_q)$
```

Figure 4: The generation algorithm Gen

The transformation has to satisfy the following properties:

Completeness. *For any (polynomial time) algorithm C, there exists a (respectively polynomial time) simulator S such that for any $i\in\mathbb{Z}$,*

$$C^{A_i} \equiv S^{B_j}(\rho),$$

where $(j,\rho)\leftarrow\mathsf{Gen}(1^n,i)$.

Security. *For any (polynomial time) algorithm C, there exists a (respectively polynomial time) time simulator S such that for any possibly mixed quantum state* aux *and for any $i\in\mathbb{Z}$,*

$$C^{B_j}(1^n,\rho\otimes\mathsf{aux}) \sim_s S^{A_i}(1^n,\mathsf{aux}),$$

where $(j,\rho)\leftarrow\mathsf{Gen}(1^n,i)$.

3.1 The transformation

Here we formally present the construction that transforms any polynomial time stateful oracle into a stateless one. Intuitively, the construction works as follows. Our new stateless oracle B has to take as input all the previous queries. In this way, we guarantee that B does not need to keep a state. On the other hand, we have to impose that B cannot be rewound, i.e., if the first query is x, then there is no way we can start B from the beginning with a query $x'\neq x$. To achieve this, B is parameterized by a list $s_1,\ldots,s_q$ of secret keys for a DMAC, where q is the total number of queries. For each query x_j, the calling algorithm has to also provide a tag t_j for x_j corresponding to the secret key s_j. Before executing the query, B first verifies that the tags for all the queries are valid. If this is the case, then it runs all the queries one by one and returns the final outcome.

Let $\mathcal{A}=\{A_i\}_{i\in\mathbb{Z}}=\{C_{i,j}\}_{i,j\in\mathbb{Z}}$ be the class of all polynomial time oracles, where $C_{i,j}$ are the stateless oracles corresponding to A_i. Moreover, let $(\mathsf{DMAC.Gen},\mathsf{DMAC.Tag},\mathsf{DMAC.Ver})$ be a secure DMAC. We define the class $\mathcal{B}=\{B_{(s_1,\ldots,s_q,i)}\}_{s_1,\ldots,s_q,i}$ in figure. 3.

Clearly, $\mathcal{B}$ is a class of stateless oracles. Now, the generation algorithm Gen works as shown in figure 4.

To argue completeness, let C be any algorithm that has access to the stateful oracle A. We will create a simulator S that takes as input the quantum state $\rho_1\otimes\ldots\otimes\rho_q$ and has oracle access to the stateless oracle B. S initializes $\tau=0$ and the sequence $\mathcal{S}$ to be the empty sequence. Then it starts C and simulates C's oracle as shown in figure 5.

```
ORACLE($x$)
    $\tau \leftarrow \tau + 1$
    $t \leftarrow \mathsf{DMAC.Tag}(\rho_\tau, x)$
    Append $(x, t)$ on $\mathcal{S}$ and parse $\mathcal{S}$ as $((x_1, t_1), \ldots, (x_\tau, t_\tau))$
    return $B((x_1, t_1), \ldots, (x_\tau, t_\tau))$
```

Figure 5: The oracle that S simulates

Therefore, the completeness follows from that of the DMAC.

3.2 Security analysis

To argue security, we create a simulator S that takes as input an auxiliary state aux and has oracle access to the algorithm A. S first creates q pairs of DMAC keys $s_1, \ldots, s_q$ together with their quantum states $\rho_1, \ldots, \rho_q$. Then S starts C with input $\rho_1 \otimes \ldots \otimes \rho_q \otimes \mathsf{aux}$. Moreover, S simulates the oracle B as shown in figure 6. During the simulation, S initializes two empty lists $\mathcal{Q}, \mathcal{A}$ whose size increases at the same time. Informally, $\mathcal{Q}$ will contain the longest sequence of queries $x_1, \ldots, x_{|\mathcal{Q}|}$ that have a valid tag. $\mathcal{A}$ will contain the corresponding answers that the algorithm A replies. We denote by $\mathcal{Q}_i$ the i-th element of $\mathcal{Q}$ and similarly for $\mathcal{A}$.

```
$B_{\mathsf{sim}}((x_1, t_1), \ldots, (x_\tau, t_\tau))$
    if $\mathsf{DMAC.Ver}(s_j, x_j, t_j) = 0$ for some $j \in [\tau]$ then
        return "Invalid tag"
    if $(x_1, \ldots, x_\tau)$ is a prefix of $\mathcal{Q}$ then
        return $\mathcal{A}_\tau$ (no need to query $A$)
    if $\mathcal{Q}$ is not a prefix of $(x_1, \ldots, x_\tau)$ then
        return $\perp$
    $l \leftarrow |\mathcal{Q}| + 1$
    for $i \in [l, \tau]$ do
        $\mathcal{Q}_i \leftarrow x_i$ (i.e. append $x_i$ to $\mathcal{Q}$)
        $\mathcal{A}_i \leftarrow A(x_i)$ (i.e. append the answer to $\mathcal{A}$)
    return $\mathcal{A}_\tau$
```

Figure 6: The oracle B_{sim} that S simulates

Note that if the execution reaches the line **return** $\perp$, then the adversary will be able to tag two messages using the same key.

Notice that if C is computationally unbounded but limited to a polynomial number of queries, then S has to be also computationally unbounded.

Let E be the event that the line **return** $\perp$ is executed. Let q' be the number of queries C makes to its oracle B and let also $\{(x_{1j}, t_{1j}), \ldots, (x_{\tau_j j}, t_{\tau_j j})\}_{j \in [q']}$ be the queries. Equivalently, this event can be defined as the event that C makes two queries with different messages in some position i and the corresponding tags are both valid:

$$\begin{aligned} E = \{\exists j, j' \in [q'], i \in [q] : & \, x_{ij} \neq x_{ij'} \\ & \wedge \mathsf{DMAC.Ver}(s_i, x_{ij}, t_{ij}) = 1 \\ & \wedge \mathsf{DMAC.Ver}(s_i, x_{ij'}, t_{ij'}) = 1\}. \end{aligned}$$

Then, our simulator works exactly as C except for the event E; i.e., for any output o, any $n \in \mathbb{Z}$ and

```
$B_{\mathsf{sim}}((x_1, t_1), \ldots, (x_\tau, t_\tau))$
    if $\mathsf{DMAC.Ver}(s_j, x_j, t_j) = 0$ for some $j \in [\tau] - \{i^*\}$ then
        return "Invalid tag"
    if $i^* \leq \tau$ and $V(x_{i^*}, t_{i^*}) = 0$ then
        return "Invalid tag"
    if $(x_1, \ldots, x_\tau)$ is a prefix of $\mathcal{Q}$ then
        return $\mathcal{A}_\tau$ (no need to query $A$)
    if $\mathcal{Q}$ is not a prefix of $(x_1, \ldots, x_\tau)$ and $x_{i^*} \neq \mathcal{Q}_{i^*}$ then
        Stop simulation and return $(x_{i^*}, t_{i^*}, \mathcal{Q}_{i^*}, t^*)$
    else
        Abort
    if $i^* \leq \tau$ then
        $t^* \leftarrow t_{i^*}$ (remember the first tag)
    $l \leftarrow |\mathcal{Q}| + 1$
    for $i \in [l, \tau]$ do
        $\mathcal{Q}_i \leftarrow x_i$ (i.e. append $x_i$ to $\mathcal{Q}$)
        $\mathcal{A}_i \leftarrow A(x_i)$ (i.e. append the answer to $\mathcal{A}$)
    return $\mathcal{A}_\tau$
```

Figure 7: The oracle B_{sim} simulated by C'.

any auxiliary quantum state aux, it holds that

$$\left|\Pr[C^{B_{(s_1,\ldots,s_q,i)}}(1^n, \rho_1 \otimes \ldots \otimes \rho_q \otimes \mathsf{aux}) = o] - \Pr[S^{A_i}(1^n, \mathsf{aux}) = o]\right| \leq \Pr[E].$$

Now, suppose that there exists an adversary C, value $n \in \mathbb{Z}$ and quantum state aux such that $\Pr[E] \geq e(n)$ for some non-negligible function e. We use C to create an adversary C' against the DMAC. C' takes as input a quantum state ρ and has oracle access to the algorithm $V(\cdot, \cdot)$. It starts by picking a random position $i^* \leftarrow [q]$. In this position, C' will plug the quantum state ρ. For simplicity we rename ρ as ρ_{i^*}. Moreover, C' creates $q-1$ pairs $(s_i, \rho_i) \leftarrow \mathsf{DMAC.Gen}(1^n)$ for $i \in [q] - \{i^*\}$. Then C' runs C with input $(\rho_1 \otimes \ldots \otimes \rho_q \otimes \mathsf{aux})$ and simulates the oracle B as shown below (fig. 7). As before C' has to keep two lists $\mathcal{Q}, \mathcal{A}$ that are initialized to the empty lists.

Informally, C' runs by simulating the stateless oracle and at the same time looking for a pair of inputs that can break the challenge DMAC. For the queries that do not correspond to i^*, C can use its own secret key. For the ones that correspond to i^*, the simulator uses its verification oracle V. If the adversary ever submits two different sequences of queries such that they are not a prefix of each other, then the simulation stops. With probability $1/q$ the sequences will differ on the i^*-th position, in which case C' will be able to break its challenge.

We can see that $\mathsf{Adv}_{C'}^{\mathsf{DMAC}} = \Pr[E]/q \geq e(n)/q$, which implies that C' breaks the DMAC game with non-negligible probability by using only polynomially many queries to the verification oracle.

4 Constructing One-time Primitives

In this section we present definitions and constructions of one-time primitives. In particular, we define one-time memories (OTMs) and one-time programs (OTPs). In Subsection 4.1 we define OTMs in the plain model and we discuss their impossibility. We then provide a definition of OTMs relative to an oracle. In Subsection 4.2 we define OTPs again relative to an oracle. In Subsection 4.3 we present a construction of OTM relative to a classical oracle. Last, in Subsection 5.1 we define "extendable OTMs," which are OTMs that can be extended to encapsulate more secrets.

4.1 One-time memories

An OTM is a memory that stores k secrets $s_1, s_2, \ldots, s_k$ but only one can be extracted. OTMs were first defined by Goldwasser and Rothblum [15] in order to construct OTPs. OTMs have then been studied extensively in the quantum setting, with Broadbent et al. [7] presenting a construction of quantum OTPs from quantum OTMs. Liu [19] has shown that OTMs are possible in the isolated qubits model, where each single qubit is manipulated by an adversary and those adversaries are allowed to communicate only classically.

Definition 5 (One-time memories)**.** *A 1-out-of-k one-time memory (OTM) scheme is a pair of algorithms* $(\mathsf{Gen}, \mathsf{Extract})$ *defined as follows:*

- $\mathsf{Gen}(1^n, s_1, s_2, \ldots, s_k) \to \rho$ *is an algorithm that takes as input k secret classical bit-strings $s_1, s_2, \ldots, s_k$, each of length n, and outputs a quantum state ρ that encodes the k secrets.*
- $\mathsf{Extract}(\rho, i) \to s$ *is an algorithm that takes as input an index i and a quantum state ρ and outputs a classical bit-string s.*

An OTM scheme satisfies the following security properties.

Completeness. *For any $s_1, s_2, \ldots, s_k \in \{0,1\}^n$ and for any index i, it holds that* $\mathsf{Extract}(\mathsf{Gen}(1^n, s_1, s_2, \ldots, s_k), i) = s_i$.

One-timeness. *For any (possibly unbounded) adversary A, there exists a (respectively unbounded) simulator S, such that for any k bit-strings $s_1, s_2, \ldots, s_k$ of length n and any auxiliary (mixed) quantum state* aux*,*

$$A(1^n, \mathsf{Gen}(1^n, s_1, s_2, \ldots, s_k) \otimes \mathsf{aux}) \sim_s S^{\mathsf{OT}_{s_1, s_2, \ldots, s_k}}(1^n, \mathsf{aux}),$$

where the distributions are over the coins of A, S, and Gen*. The oracle $\mathsf{OT}_{s_1, s_2, \ldots, s_k}$ on input i returns s_i and then halts.*

By a slight abuse of terminology, we at times refer to $\rho \leftarrow \mathsf{Gen}(1^n, s_1, s_2, \ldots, s_k)$ as *an OTM with contents* $s_1, s_2, \ldots, s_k$.

It is well known that it is impossible to achieve OTM even in the quantum world and even with computational assumptions.

(Classical) oracle OTMs. A possible way to bypass the above impossibility is allowing OTMs classical oracle access to a classical algorithm. Our definition below is inspired by the definition of Broadbent et al. [6].

Definition 6 (Classical-oracle one-time memories)**.** *Let $\mathcal{C} = \{C_j\}_{j \in \{0,1\}^*}$ be a family of polynomial-sized classical circuits. A $\mathcal{C}$-oracle one-time memory ($\mathcal{C}$-OTM) scheme is a pair of polynomial-sized quantum algorithms* $(\mathsf{Gen}, \mathsf{Extract})$ *with the following properties:*

- $\mathsf{Gen}(1^n, s_1, s_2, \ldots, s_k) \to (\rho, j)$ *is an algorithm that takes as input k secret classical bit-strings $s_1, s_2, \ldots, s_k$, each of length n, and outputs a quantum state ρ that intuitively encodes the k secrets. In addition, it outputs an index j corresponding to the circuit C_j.*
- $\mathsf{Extract}^C(\rho, i) \to s$ *is an oracle algorithm that takes as input an index i and a quantum state ρ and makes a single oracle query to a circuit C. It outputs a classical bit-string s.*

A $\mathcal{C}$-OTM satisfies the following security properties.

Completeness. *For any $s_1, s_2, \ldots, s_k \in \{0,1\}^n$, for any index i and for any (ρ, j) that is output by* $\mathsf{Gen}(1^n, s_1, s_2, \ldots, s_k)$*, it holds that* $\mathsf{Extract}^{C_j}(\rho, i) = s_i$.

One-timeness. *For any (possibly unbounded) adversary A there exists a (respectively unbounded) simulator S, such that for any k bit-strings $s_1, s_2, \ldots, s_k$ of length n and for any auxiliary (mixed) quantum state* aux,

$$A^{C_j}(1^n, \rho \otimes \mathsf{aux}) \sim_s S^{\mathsf{OT}_{s_1, s_2, \ldots, s_k}}(1^n, \mathsf{aux}),$$

where $(\rho, j) \leftarrow \mathsf{Gen}(1^n, s_1, s_2, \ldots, s_k)$ and the distributions are over the coins of A, S, and Gen*. The oracle* $\mathsf{OT}_{s_1, s_2, \ldots, s_k}$ *on input i returns s_i and then halts.*

Similarly to the original terminology, we will refer to ρ derived as above, *as a $\mathcal{C}$-OTM with contents $s_1, s_2, \ldots, s_k$ and oracle $C_j \in \mathcal{C}$.* We will also use $\mathsf{OTM}(s_1, s_2, \ldots, s_k)$ to denote the corresponding pair (ρ, C_j).

Notice that this primitive is impossible to achieve in the classical world. Indeed, an adversary A with oracle access to C, given a classical bitstring ρ, can do the following trivial attack. First, copy the ρ into a new register ρ'. Then, run $\mathsf{Extract}^C(\rho, 0)$ and $\mathsf{Extract}^C(\rho', 1)$ and return the two results. Clearly, this cannot be simulated by S.

On the other hand, currently it is not clear whether this primitive is achievable in the quantum world. Now A cannot simply undo the computation since the oracle C works only classically and A has only oracle access to it.

Remark: The above definition seems to be the most natural way to model classical stateless tokens. In practice the above definition can be instantiated by having a programmable trusted hardware that runs some code based on some secret information. For example, smartcards or Intel's SGX could be a possible way to achieve such a hardware. Notice that obfuscating such an oracle and giving it to the adversary is not secure: since the adversary is given actual (albeit obfuscated) code in the form of a description of a quantum circuit with elementary quantum gates, it is always possible to reverse the gates and thus the whole computation.

4.2 One-Time Programs

We next define OTPs [15, 7]. As they are a generalization of OTMs, they are also impossible in the quantum plain model.

(Classical) oracle one-time programs. In the following we define oracle OTPs, which are programs that can be run exactly once. To overcome the impossibility, we allow OTPs access to a classical oracle with classical interface. Our definition is inspired by the definition of Broadbent et al. [6].

Definition 7 (Classical Oracle one-time Programs)**.** *Let $\mathcal{C} = \{C_j\}_{j \in \{0,1\}^*}, \mathcal{C}' = \{C'_{j'}\}_{j' \in \{0,1\}^*}$ be two classes of polynomial-sized classical circuits. A $(\mathcal{C}, \mathcal{C}')-$one-time program (denoted as $(\mathcal{C}, \mathcal{C}')-$OTP) is a pair of algorithms* $(\mathsf{Gen}, \mathsf{Extract})$ *with the following properties:*

- $\mathsf{Gen}(1^n, j) \to (\rho, j')$ *is an algorithm that takes as input a security parameter n and an index j and outputs a quantum state ρ and an index j'.*
- $\mathsf{Extract}^C(\rho, x) \to y$ *is a quantum algorithm that takes as input a quantum state ρ and a classical input x and has oracle access to a circuit C. It outputs a bit-string y.*

An OTP satisfies the following two properties.

Completeness. *For any $n \in Z$, any $j \in \{0,1\}^n$ and any input x, it holds that*

$$\mathsf{Extract}^{C'_{j'}}(\rho, x) = C_j(x),$$

where $(\rho, j') \leftarrow \mathsf{Gen}(1^n, j)$.

Security. *For any (possibly unbounded) adversary A, there exists a (respectively unbounded) simulator S, such that for any $n \in \mathbb{Z}$, any $j \in \{0,1\}^n$ and any quantum auxiliary (mixed) quantum state* aux,

$$A^{C'_{j'}}(1^n, \rho \otimes \mathsf{aux}) \sim_s S^{\mathsf{OT}_j}(1^n, \mathsf{aux}),$$

where $(\rho, j') \leftarrow \mathsf{Gen}(1^n, j)$ *and the distributions are over the coins of A, S, and* Gen. *The oracle* OT_j *on input x returns $C_j(x)$ and then halts.*

4.3 One-time program construction

Using our general transformation of stateful to stateless oracles, it is easy to create OTPs relative to a classical oracle. To see this, notice that any algorithm with oracle access to an OTP can be turned into an algorithm that has oracle access to a stateless version of an OTP together with a quantum state.

In the following we formally prove this idea. The reader can feel free to skip the formal proof without missing important details.

Let $\mathcal{C} = \{C_i\}$ be the class of polynomial-sized classical circuits. Moreover, let OT_i be the one-time version of the circuit C_i, i.e., OT_i on input x returns $C_i(x)$ and then halts. Let Gen' be the transformation from Section 3 with respect to the class $\{\mathsf{OT}_i\}_i$. In other words, Gen' turns the class of stateful algorithms $\{\mathsf{OT}_i\}_i$ into the class of stateless algorithms $\mathcal{B} = \{B_j\}_j$. Let ρ be the quantum state output by Gen'. Let $D_x^{\mathsf{OT}_i}$ be an algorithm with oracle access to OT_i that returns $\mathsf{OT}_i(x)$. In other words, D_x queries its oracle OT_i with the value x and returns the result. By the completeness of the stateful-to-stateless transformation, there exists an algorithm S_x such that

$$D_x^{\mathsf{OT}_i} \equiv S_x^{B_j}(\rho),$$

where $(j, \rho) \leftarrow \mathsf{Gen}'(1^n, i)$. Our goal is to create a $(\mathcal{C}, \mathcal{B})-$OTP $(\mathsf{Gen}, \mathsf{Extract})$. We define the two algorithms in the following (fig. 8):

```
$\mathsf{Gen}(1^n, i)$
  **return** $\mathsf{Gen}'(1^n, i)$
$\mathsf{Extract}^B(\rho, x)$
  **return** $S_x^B(\rho)$
```

Figure 8: One-time Program construction

Notice that the simulator S_x with oracle access to B_j indeed returns the value $C_i(x)$, and hence the completeness follows. One-timeness follows directly from the security of the stateful-to-stateless transformation. For any adversary A there exists a simulator S' such that for any auxiliary quantum state aux, it holds that

$$A^{B_i}(1^n, \rho \otimes \mathsf{aux}) \sim_s S'^{\mathsf{OT}_i}(1^n, \mathsf{aux}).$$

4.4 Many-time Primitives

The definitions above trivially extend to the many-time cases. However, the corresponding constructions are not trivial. In other words, we cannot just create an n-time primitive by outputting n one-time primitive. To see this, consider the case of multi-bit DMACs. If we have two copies of the quantum state for tagging each bit, we can tag both 0 and 1 for each position of a bit-string, and this makes it trivial to tag any bit-string. To overcome this we use again our stateful to stateless oracle reduction. In other words, we can consider a stateful program that allows only n runs and then turn it into a stateless one. Since n-time memories and n-time message authentication codes are a special case of an n-time program, we get the corresponding primitives. In order to keep our constructions easy to read, we restrict this paper to the one-time versions.

5 Applications

In this section we give applications of our oracle one-time (i.e., dispensable backdoors) primitives. In Section 5.1 we demonstrate how one-time backdoors can resolve the smartphone conundrum of privacy vs. law enforcement, thus addressing the original problem that motivated this paper. Subsequently, in Section 5.2 we show how to extend the one-time backdoor paradigm to standard cryptographic primitives. Concretely, we look at the case of public-key encryption schemes with dispensable decryption-backdoors and we show how we can construct them. We believe that cryptography with dispensable backdoors can also be generalized to most cryptographic primitives with game-based security definition. We believe it is interesting to consider such an extension and investigate further applications/implications.

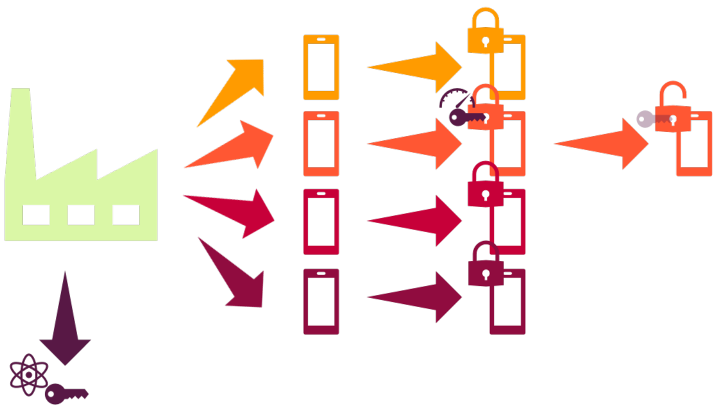

Figure 9: One-time quantum unlock key. Unlocking a phone requires measuring the state and thus collapsing the key which results in one-time use.

5.1 One-time Backdoored Devices

The original motivation was to create a system that allows a device (e.g., smartphone) vendor to embed in its devices a content locking mechanism, and create (and locally store) a dispensable unlocking backdoor. The system should satisfy the following properties:

Setup. There should be a setup algorithm that creates the code for the locking device and the relevant unlock backdoor.

Confidentiality. No one (in particular, no PPT adversary) should be able to extract any information from the locked device without the backdoor. This should be true even if the adversary has (partial) knowledge about the keys and/or states of the unlocked devices and about the state of the locked devices.

One-time unlock. Using the unlocking backdoor, the vendor should be able to unlock exactly one phone. In particular, it should not be able to use the backdoor to extract information from two locked devices. This should, again, be true even if the adversary has (partial) knowledge about the keys and/or states of the unlocked devices and about the state of the locked devices.

(Non-interactive) Extendibility. The vendor should be able to program more (new) devices to be unlockable with the dispensable backdoor without resetting the entire system, or, in particular, interacting with the devices that are already set up and distributed.

The above can be achieved by having a quantumly enabled vendor equipped with a stateless token and a classical set of devices. Note that we assume classical devices as it is unlikely that current technology will yield hand-held devices with quantum storage and computation capabilities any time soon. In addition, we do not assume that each device has a secure storage or trusted-hardware module.

Our system works as follows (see also Figure 9 for an illustration):

The Setup Algorithm: Let N be the number of initial devices, denoted by $D_1, \ldots, D_N$ that the vendor wishes to set up, and $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ denote the key-generation, encryption, and decryption algorithms for a symmetric-key encryption scheme. Without loss of generality, we will assume the scheme to be IND-CPA-secure [4] as this will already provide us with the desirable confidentiality. Of course one can consider schemes with higher level of security, e.g., IND-CCA-security, if an application needs additional security guarantees. The vendor V performs the following steps to set up all the N devices:

1. The vendor V uses the key generation algorithm Gen N times to generate N independent n-bit secret keys $k_1, \ldots, k_N$ (where n is the security parameter).

2. The code of each D_i contains the following locking procedure: D_i has the key k_i locally stored; to lock itself—e.g., if its user inputs incorrect pins too many times—D_i uses Enc to encrypt its state with key k_i and erases the key k_i. Without loss of generality, we assume that in the locked state, the phone might accept a command to output its encrypted state.[2]

[2]It is assumed that the vendor can extract the encrypted state from the phone's storage, anyway.

3. V creates a 1-out-of-N OTM that encodes the keys $k_1, \ldots, k_N$. Subsequently, the vendor erases the keys $k_1, \ldots, k_N$ from its local state (so that they only reside in the OTM) and also the coins used in their generation. Notice that, after the phone locks itself, the key is only available through the OTM, and even the vendor cannot extract the encryption key.

Unlocking a Device with the backdoor: The vendor (or anyone in possession of the OTM) can use the OTM to unlock any locked device in a straightforward manner: To unlock D_i, the vendor extracts the key k_i from the OTM and uses it to decrypt the state.

One can easily confirm the security of OTMs and the encryption scheme ensures that our protocol satisfies the properties required above, i.e., setup, confidentiality, and the one-time-unlock property: The fact that the setup algorithm achieves the setup guarantees follows directly by inspection of the protocol. Confidentiality follows directly from the CPA-security of the encryption scheme. Finally, the one-time-unlock property follows from the CPA-security of the encryption scheme and the security of the OTMs. Note that OTMs are assumed secure even with respect to any auxiliary information. Hence, (partial) knowledge about the state/keys of the unlocked devices or the state of the locked devices does not help the adversary to learn any information from the OTM about any key (or about the corresponding encrypted states) other than the extracted.

To complete our analysis, we need to describe how to obtain non-interactive extendibility. In order to do this, we define in the following an extendable version of OTMs, which are memories that can be extended by adding more secrets into them. Using such memories, one can trivially add new devices in the system without interacting with existing devices by simply running the setup algorithm for the new devices and adding the new keys to the existing OTM, instead of storing them in a new OTM.

Extendable OTMs Informally, extendable OTMs are OTMs that can be encapsulated with additional secrets. Correctness should guarantee that any of the up-to-now secrets encoded can be extracted, whereas security should guaranteed that only one of these secrets can be extracted. In this section we will omit the oracles in notation to simplify the presentation.

Definition 8 (Extendable one-time memories)**.** *An extendable one-time memory (EOTM) is an OTM (as in Definition 5) augmented with an extra algorithm* Extend *as follows:*

- $\mathsf{Extend}(1^n, \rho, s) \to \rho'$ *is an algorithm that takes as input a quantum state ρ encoding some secrets and a classical bit-string s of length n, and returns a quantum state ρ' encoding the previous secrets plus s.*

The completeness and security extend trivially from the definitions of OTMs. In particular, in this case we want that for any algorithm A that takes as input an EOTM encoding k secrets $s_1, \ldots, s_k$ to have a simulator that can compute anything that A can compute but with only oracle access to $\mathsf{OT}_{s_1,\ldots,s_k}$, where $\mathsf{OT}_{s_1,\ldots,s_k}$ takes as input an index i, returns s_i and then halts.

5.1.1 A black-box construction of EOTMs

Our construction turns any OTM into an EOTM in a black-box manner. We give the intuition with an example below that illustrates how to go from an OTM that encodes two secrets (the first line) to one that encodes three secrets (the second line). We denote by $\mathsf{OTM}(s_0, s_1)$ the outcome of an one-out-of-two OTM generation algorithm with input s_0, s_1.

$$\mathsf{OTM}(s_1, k_1), \mathsf{OTM}(s_2 \oplus k_1, k_2), k_1 \oplus k_2$$
$$\mathsf{OTM}(s_1, k_1), \mathsf{OTM}(s_2 \oplus k_1, k_2), \mathsf{OTM}(s_3 \oplus k_1 \oplus k_2, k_3), k_1 \oplus k_2 \oplus k_3$$

One can see that we use the classical value of an OTM as a mask for the next secret and then we update the classical value by XORing it with a new key. Notice that in order to extract the value, say s_3, we need to extract both k_1 and k_2 from the previous OTMs and thus we do not have the option to extract any of the other secrets.

Formally, let $(\mathsf{OTM.Gen}, \mathsf{OTM.Extract})$ be an one-out-of-two OTM. In the following we omit the security parameter input 1^n for simplicity. We create an EOTM $(\mathsf{Gen}, \mathsf{Extend}, \mathsf{Extract})$ as follows (fig. 10):

```
Gen($s_1, s_2$)
    $k_1, k_2 \leftarrow \{0,1\}^n$
    return $\mathsf{OTM.Gen}(s_1, k_1) \otimes \mathsf{OTM.Gen}(s_2 \oplus k_1, k_2), k_1 \oplus k_2$
Extend($(\rho, k), s$)
    $k' \leftarrow \{0,1\}^n$
    return $\rho \otimes \mathsf{OTM.Gen}(s \oplus k, k'), k \oplus k'$
Extract($(\rho_1 \otimes \ldots \otimes \rho_i, \_), j$)
    $k \leftarrow 0^n$
    for $l = 1$ to $j - 1$ do
        $k \leftarrow k \oplus \mathsf{OTM.Extract}(\rho_l, 1)$
    return $\mathsf{OTM.Extract}(\rho_j, 0) \oplus k$
```

Figure 10: Extendable OTM construction

Theorem 2. *The construction above is an extendable OTM.*

Proof. Any algorithm with access to the OTM and an auxiliary input aux can be simulated, using a straightforward hybrid argument, by one algorithm that has oracle access to the respective one-time OT oracles and is also given aux as input. Moreover, any such algorithm S can be easily simulated by an algorithm S' that has oracle access to the algorithm $\mathsf{OT}_{s_1,\ldots,s_n}$ (that on input i returns s_i and then halts) as follows. S' on input aux, starts S with input aux. If S queries oracle i with bit $b = 1$, S' returns a random key k_i. If S queries oracle i with bit $b = 0$, then S' queries its oracle with value i. Upon getting answer s_i, it returns the value $s_i \oplus (\bigoplus_{j<i} k_j)$, by fixing at random all the keys k_j, $j < i$ that have not been queried. $\square$

Note that it is mandatory for the previous values of k to be erased and only the final one is kept. Indeed, if an adversary has continuous access to the previous classical values of the OTM, it can retrieve all the classical keys $k_1, \ldots, k_\ell$ without destroying the OTMs and thus it can retrieve all the secrets by always extracting the first part of the OTM and XORing it with the respective secret.

5.2 Encryption with Dispensable Decryption backdoor

Our goal in this section is to turn any CCA encryption scheme into a one-time backdoored one in the sense that one can have access to a quantum backdoor that can be used only once to perform decryptions.

First, we give a general encryption definition. We then extend this definition to its one-time backdoored version. The backdoored version should satisfy three properties. First, it should satisfy the correctness of the original scheme. Second, it should satisfy the correctness of the backdoor, i.e., the backdoor correctly decrypts a ciphertext. Last, in the security game, the adversary is given additionally a quantum backdoor and its goal is to now break *two* challenges.

Definition 9 (Encryption)**.** *An encryption scheme consists of three algorithms $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with the following properties:*

- $\mathsf{Gen}(1^n) \rightarrow (\mathsf{ek}, \mathsf{dk})$ *is a key-generation algorithm that takes as input a security parameter n and returns an encryption key and a decryption key. In the case of symmetric encryption* $\mathsf{dk} = \mathsf{ek}$.
- $\mathsf{Enc}(\mathsf{ek}, m) \rightarrow c$ *is the encryption algorithm that takes as input the encryption key and a message m and returns a ciphertext c.*
- $\mathsf{Dec}(\mathsf{dk}, c) \rightarrow m$ *is the decryption algorithm that takes as input the decryption key and a ciphertext and returns a message m or $\perp$.*

Completeness. *E is complete if*

$$\Pr[\mathsf{Dec}(\mathsf{dk}, \mathsf{Enc}(\mathsf{ek}, m)) = m] = 1,$$

where the randomness is over $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Gen}(1^n)$.

Security is defined via a code-based game G [5] between a challenger and an adversary A, where A is given access to some oracles as shown below. The adversary begins by calling the oracle INIT, which returns an encryption key. Then the adversary is allowed to call the oracles ENC, DEC and CHAL and in the end it calls the oracle FIN, which finally outputs a bit. For an adversary A, let $\mathsf{Adv}_A^{\mathsf{CCA}} = \left|\Pr[\text{FIN} = 1] - \frac{1}{2}\right|$ for the security game defined below (fig. 11).

Security. *E is secure if for any polynomial time quantum adversary A it holds that*

$$\mathsf{Adv}_A^{\mathsf{CCA}} \leq \mathsf{negl}(n).$$

```
ORACLE INIT($1^n$)
    $b \leftarrow \{0,1\}$
    $S \leftarrow \emptyset$
    $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Gen}(1^n)$
    return $\mathsf{ek}$
ORACLE ENC($m$)
    return $\mathsf{Enc}(\mathsf{ek}, m)$
ORACLE CHAL($m_0, m_1$)
    $c^* \leftarrow \mathsf{Enc}(\mathsf{ek}, m_b)$
    return $c^*$
ORACLE DEC($c$)
    $S \leftarrow S \cup \{c\}$
    return $\mathsf{Dec}(\mathsf{dk}, c)$
ORACLE FIN($b'$)
    if $c^* \notin S$ then
        return $b = b'$
    else
        return $\perp$
```

Figure 11: Standard CCA security

The above definition captures both public-key and secret-key encryption. In the case of secret-key encryption, the oracle ENC is not redundant. A more general definition that captures both public-key encryption and identity-based encryption is also possible. Moreover, it could be possible to abstract the definition even more in order to capture signatures or ideally any cryptographic primitives. We reserve such a definition and its corresponding one-time backdoor construction for a future work.

Encryption with dispensable backdoors. An encryption scheme with a dispensable backdoor is an encryption scheme augmented so that the generation algorithm produces also a quantum backdoor β as well as a description of a classical oracle. Moreover, the scheme has an additional algorithm Rec that uses the backdoor to decrypt a ciphertext.

Definition 10 (Encryption with dispensable backdoors)**.** *Let $\mathcal{C} = \{C_k\}_{k\in\{0,1\}^*}$ be a family of polynomial-sized classical circuits. A $\mathcal{C}$-backdoored encryption scheme ($\mathcal{C}$-Back) consists of algorithms* $(\mathsf{Gen}, \mathsf{Rec}, \mathsf{Enc}, \mathsf{Dec})$ *with the following properties:*

- $\mathsf{Gen}(1^n) \rightarrow (\mathsf{ek}, \mathsf{dk}, \beta, k)$ *is a key-generation algorithm that takes as input a security parameter n and returns an encryption-key* ek*, a decryption-key* dk*, a quantum state β, and an index k to a circuit C_k.*
- $\mathsf{Rec}^C(\beta, c) \rightarrow m$ *is a "one-time decryption" algorithm that takes as input a backdoor β and a ciphertext c and returns a message m or $\perp$.* Rec *has also oracle access to a classical circuit C.*

```
ORACLE INIT($1^n$)
    $b_0, b_1 \leftarrow \{0,1\}$
    $S \leftarrow \emptyset$
    $(\mathsf{ek}, \mathsf{dk}, \beta, k) \leftarrow \mathsf{Gen}(1^n)$
    return $(\mathsf{ek}, \beta)$
ORACLE ENC($m$)
    return $\mathsf{Enc}(\mathsf{ek}, m)$
ORACLE CHAL0($m_0, m_1$)
    $c_0^* \leftarrow \mathsf{Enc}(\mathsf{ek}, m_{b_0})$
    return $c_0^*$
ORACLE CHAL1($m_0, m_1$)
    $c_1^* \leftarrow \mathsf{Enc}(\mathsf{ek}, m_{b_1})$
    return $c_1^*$
ORACLE DEC($c$)
    $S \leftarrow S \cup \{c\}$
    return $\mathsf{Dec}(\mathsf{dk}, c)$
ORACLE C($x$)
    return $C_k(x)$
ORACLE FIN($b_0', b_1'$)
    if $c_0^* \notin S$ and $c_1^* \notin S$ then
        return $b_0 = b_0'$ and $b_1 = b_1'$
    else
        return $\perp$
```

Figure 12: One-time backdoored CCA security

- $\mathsf{Enc}(\mathsf{ek}, m) \to c$ *is the encryption algorithm that takes as input the encryption key and a message* m *and returns a ciphertext* c.
- $\mathsf{Dec}(\mathsf{dk}, c) \to m$ *is the decryption algorithm that takes as input the decryption key and a ciphertext and returns a message* m *or* $\perp$.

Completeness. $\mathcal{C}$-Back *is complete if*

$$\Pr_{(\mathsf{ek},\mathsf{dk},\beta,k)\leftarrow\mathsf{Gen}(1^n)}[\mathsf{Dec}(\mathsf{dk}, \mathsf{Enc}(\mathsf{ek}, m)) = m] = 1,$$

and moreover,

$$\Pr_{(\mathsf{ek},\mathsf{dk},\beta,k)\leftarrow\mathsf{Gen}(1^n)}[\mathsf{Rec}^{C_k}(\beta, \mathsf{Enc}(\mathsf{ek}, m)) = m] = 1.$$

The security game is similar to the original one, with the additional property that the adversary is given one backdoor and has to break two challenges as shown in the game below. Notice that an adversary, who just uses the backdoor to decrypt one challenge and then guesses the other, has a probability of 1/2 to win. Moreover, notice the adversary is also given access to the classical oracle C_k. For an adversary A, let $\mathsf{Adv}_A^{\mathsf{BCCA}} := \left|\Pr[\text{FIN} = 1] - \frac{1}{2}\right|$ in the security game defined below (fig. 12).

Security. $\mathcal{C}$-Back *is secure if for any polynomial time quantum adversary* A *it holds that*

$$\mathsf{Adv}_A^{\mathsf{BCCA}} \leq \mathsf{negl}(n).$$

```
$\mathsf{Gen}(1^n)$
    $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Gen}'(1^n)$
    $(\beta, k) \leftarrow \mathsf{OTP.Gen}(\mathsf{dk})$
    return $(\mathsf{ek}, \mathsf{dk}, \beta, k)$
$\mathsf{Rec}^C(\beta, m)$
    return $\mathsf{OTP.Extract}^C(\beta, m)$
$\mathsf{Enc}(\mathsf{ek}, m)$
    return $\mathsf{Enc}'(\mathsf{ek}, m)$
$\mathsf{Dec}(\mathsf{dk}, c)$
    return $\mathsf{Dec}'(\mathsf{dk}, c)$
```

Figure 13: Encryption with Dispensable Backdoor construction

5.3 Constructing Encryption with Dispensable Backdoor

Here we show how to construct Encryption with Dispensable Backdoor using oracle OTPs. As we have shown above, oracle OTPs are possible in the plain model. The idea is to use the OTP generation algorithm with input the description of the decryption algorithm. In other words, all we have to do is to create a one-time version of the decryption algorithm and this will be our backdoor. Let $G' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ be an IND-CCA secure encryption scheme. Let $\mathcal{D} = \{D_{\mathsf{dk}}\}_{\mathsf{dk}}$ be the class of polynomial-sized circuits such that $D_{\mathsf{dk}}(m) = \mathsf{Dec}'(\mathsf{dk}, m)$. We have shown above how to construct OTPs for any class of polynomial-sized circuits, and thus, in particular for $\mathcal{D}$. Therefore, let $(\mathsf{OTP.Gen}, \mathsf{OTP.Extract})$ be a $(\mathcal{D}, \mathcal{C})-$OTP for some class of circuits $\mathcal{C} = \{C_k\}_k$. We create a $\mathcal{C}$-backdoored encryption scheme $G = (\mathsf{Gen}, \mathsf{Rec}, \mathsf{Enc}, \mathsf{Dec})$ as follows (fig.13):

Theorem 3. *G is a $\mathcal{C}-$backdoored encryption scheme.*

Proof. The two completeness properties are trivially satisfied by invoking the completeness property of the original encryption scheme and the completeness property of the OTP.

To argue security, note that an adversary A with a backdoor β can be simulated by a simulator S who has access to an additional stateful oracle that decrypts only once and does not add this ciphertext to the set of queried ciphertexts. In this step, the decryption key dk is considered as the auxiliary state. Call $G1$ the game played by S. Now an adversary S who can win this game can easily be turned into an adversary A' that breaks CCA. A' simulates S's oracles $\textsc{Init}$,$\textsc{Enc}$,$\textsc{Dec}$ by calling its own oracles. A' will pick a random bit b and when S calls its oracle $\textsc{Chal}(b)$ with messages m_0, m_1, A' will use its own challenge oracle with input m_0, m_1. It will get a ciphertext c_b^* and will forward this answer to S. When S calls its oracle $\textsc{Chal}(1-b)$ then A' will encrypt at random one of the two messages using its own encryption oracle; call this ciphertext c_{1-b}^*. When S calls its one-time decryption oracle, there are three cases. If S queries c_b^*, then A' will reply either m_0 or m_1 with probability 1/2. If S queries c_{1-b}^*, then A' will reply with the corresponding plaintext since A' knows which message it corresponds to. If S queries any other ciphertext, then A' will use its own decryption oracle. Since A' picks the bit b at random, there is at most 1/2 probability that S will not query its one-time decryption oracle with the ciphertext c_b^*. Finally, when S makes a guess between m_0, m_1, A' will return the same guess.

Suppose that there exists a non-negligible function $e(n)$ such that

$$\mathsf{Adv}_A^{\mathsf{BCCA}} \geq e(n).$$

Then by the security of OTP, for the advantage Adv_S^{G1} of S to win the modified game $G1$, it holds that

$$\mathsf{Adv}_S^{G1} \geq e(n) - \mathsf{negl}(n).$$

Thus the advantage of A' in the CCA game is

$$\mathsf{Adv}_{A'}^{\mathsf{CCA}} \geq \frac{1}{2} \cdot \mathsf{Adv}_S^{G1} \geq \frac{1}{2} \cdot e(n) - \mathsf{negl}(n),$$

which is non-negligible.

References

[1] Scott Aaronson. Limitations of quantum advice and one-way communication. In *Computational Complexity, 2004. Proceedings. 19th IEEE Annual Conference on*, pages 320–332. IEEE, 2004.

[2] Scott Aaronson and Paul Christiano. Quantum money from hidden subspaces. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 41–60, New York, NY, USA, 2012. ACM.

[3] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Annual International Cryptology Conference*, pages 1–18. Springer, 2001.

[4] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, pages 26–45, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[5] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 409–426, 2006.

[6] Anne Broadbent, Sevag Gharibian, and Hong-Sheng Zhou. Quantum one-time memories from stateless hardware. Cryptology ePrint Archive, Report 2015/1072, 2015.

[7] Anne Broadbent, Gus Gutoski, and Douglas Stebila. Quantum one-time programs - (extended abstract). In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 344–360, 2013.

[8] Ran Canetti, Guy N Rothblum, and Mayank Varia. Obfuscation of hyperplane membership. In *TCC*, volume 5978, pages 72–89. Springer, 2010.

[9] James B. Comey. Transcripts from a public speech, Brookings Institution, Washington, D.C. `https://www.fbi.gov/news/speeches/going-dark-are-technology-privacy-and-public-safety-on-a-collision-course`.

[10] Shalev Ben David and Or Sattath. Quantum tokens for digital signatures. *arXiv preprint arXiv:1609.09047*, 2016.

[11] Electronic Frontier Foundation (EFF). The crypto wars: Governments working to undermine encryption. Technical Report, 2014. `https://www.eff.org/files/2014/01/03/cryptowarsonepagers-1_cac.pdf`.

[12] Dmitry Gavinsky. Quantum money with classical verification. In *Computational Complexity (CCC), 2012 IEEE 27th Annual Conference on*, pages 42–52. IEEE, 2012.

[13] Marios Georgiou and Iordanis Kerenidis. New constructions for quantum money. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 44. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[14] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications.* Cambridge university press, 2009.

[15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. One-time programs. In *Annual International Cryptology Conference*, pages 39–56. Springer, 2008.

[16] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. *IACR Cryptology ePrint Archive*, 2017:274, 2017.

[18] Arash Khamooshi. Breaking down Apple's iPhone fight with the U.S. Government. The New York Times, 2016. `https://www.nytimes.com/interactive/2016/03/03/technology/apple-iphone-fbi-fight-explained.html?_r=0`.

[19] Yi-Kai Liu. Single-shot security for one-time memories in the isolated qubits model. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 19–36, 2014.

[20] Fernando Pastawski, Norman Y Yao, Liang Jiang, Mikhail D Lukin, and J Ignacio Cirac. Unforgeable noise-tolerant quantum tokens. *Proceedings of the National Academy of Sciences*, 109(40):16079–16082, 2012.

[21] Michael S. Schmidt and Richard Perez-Pena. F.B.I. treating san bernardino attack as terrorism case. The New York Times, 2015. `https://www.nytimes.com/2015/12/05/us/tashfeen-malik-islamic-state.html`.

[22] Bruce Schneider. More crypto wars ii. Blog: Schneider on Security, 2014. `https://www.schneier.com/blog/archives/2014/10/more\_crypto\_war.html`.

[23] Hoeteck Wee. On obfuscating point functions. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 523–532. ACM, 2005.

[24] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under lwe. *IACR Cryptology ePrint Archive*, 2017:276, 2017.

6 Appendix: DMAC construction

Here we include the construction of dispensable message-authentication codes as defined implicitly by David and Sattath [10] which builds upon the construction of Aaronson and Christiano [2]. Let H be the n-qubit Hadamard operator. For a subspace $A \leq \mathbb{F}_2^n$, let $A^\perp$ be its orthogonal complement.

```
Gen($1^n$)
    Pick a random subspace $A \leq \mathbb{F}_2^n$ of dimension $n/2$
    $\rho = \frac{1}{\sqrt{|A|}} \sum_{\mathbf{v} \in A} |v\rangle$
    return $(A, \rho)$
Tag($\rho, b$)
    if $b = 0$ then
        Measure $\rho$ and return the outcome
    else
        Measure $H \rho H^\intercal$ and return the outcome
Ver($A, b, \mathbf{v}$)
    if $b = 0$ then
        return $\mathbf{v} \in^? A$
    else
        return $\mathbf{v} \in^? A^\perp$
```

The completeness of the scheme follows easily from [2, Lemma 21]. The security of the scheme follows directly from [10, Theorem 16]. In particular, it is proven that any (even computationally unbounded) quantum adversary that is given as input ρ, needs an exponential number of queries to the verification oracle in order to find a vector in A and a vector in $A^{\perp}$.