One-Time Programs made Practical

Lianying Zhao[1], Joseph I. Choi[2], Didem Demirag[3], Kevin R. B. Butler[2], Mohammad Mannan[3], Erman Ayday[4], and Jeremy Clark[3]

[1] University of Toronto, Toronto ON, Canada
[2] University of Florida, Gainesville FL, USA
[3] Concordia University, Montreal QC, Canada
[4] Case Western Reserve University, Cleveland OH, USA

Abstract. A one-time program (OTP) works as follows: Alice provides Bob with the implementation of some function. Bob can have the function evaluated exclusively on a single input of his choosing. Once executed, the program will fail to evaluate on any other input. State-of-the-art one-time programs have remained theoretical, requiring custom hardware that is cost-ineffective/unavailable, or confined to ad-hoc/unrealistic assumptions. To bridge this gap, we explore how the Trusted Execution Environment (TEE) of modern CPUs can realize the OTP functionality. Specifically, we build two flavours of such a system: in the first, the TEE directly enforces the one-timeness of the program; in the second, the program is represented with a garbled circuit and the TEE ensures Bob's input can only be wired into the circuit once, equivalent to a smaller cryptographic primitive called one-time memory. These have different performance profiles: the first is best when Alice's input is small and Bob's is large, and the second for the converse.

1 Introduction

Consider the well-studied scenario of secure two-party computation: Alice and Bob want to compute a function on their inputs, but they do not want to disclose these inputs to each other (beyond what can be inferred from the output of the computation). This is traditionally handled by an interactive protocol between Alice and Bob.[5] In this paper, we instead study a non-interactive protocol as follows: Alice prepares a device for Bob with the function and her input included; once Bob receives this device from Alice, he supplies his input and learns the outcome of the computation. The device will not reveal the outcome for any additional inputs (thus, a one-time program [17]). Alice might be a company selling the device in a retail store, and Bob the customer; the two never interact directly. By using the device offline, Bob is assured that his input remains private.

To build a one-time program (OTP), we use the Trusted Execution Environment (TEE), a hardware-assisted secure mode on modern processors, where execution integrity and secrecy are ensured [43], with qualities that include platform state binding and protection of succinct secrets.[6] TEEs may appear to offer

[5] Hazay and Lindell [24] give a thorough treatment of interactive two-party protocols.
[6] Further explanation is provided in Appendix A.2.

a trivial solution to OTPs; however, complexities arise due to Bob's physical possession of the device and, more importantly, performance issues. We propose two configurations for one-time programs built on TEEs: (1) deployed directly in the TEE, and (2) deployed indirectly via TEE-backed one-time memory (OTM) [17] and garbled circuits [71] outside of the TEE. OTMs hold two keys, only one of which gets revealed (dependent on its input); the other is effectively destroyed.

Contributions. Our system, built using Intel Trusted Execution Technology (TXT) [18] and Trusted Platform Module (TPM) [62] as the TEE, is available today (as opposed to custom OTP/OTM implementations using FPGA [29], PUF [33], quantum mechanisms [8], or online services [34]) and could be built for less than \$500.[7]

We propose and implement the following OTP variants, considering that TPM-sealing[8] or encrypting data is time-consuming.

- *TXT-only* seals/unseals Alice's input directly, and performance is thus sensitive to Alice's input size. Bob's input is entered in plaintext and processed in TXT after he has received the device.
- *GC-based* converts the logic into garbled circuit, where number of key pairs is determined by Bob's input size. Key pairs are encrypted/decrypted with a master key (MK). This way, the performance is largely determined by Bob's input size. Upon receiving the device, he does the one-time selection of key pairs in TXT to reflect his input. Thereafter, evaluation of the garbled circuit can be done on any machine with the selected keys.

To illustrate the generality of our solution, we also map the following application into our proposed OTP paradigm: a company selling devices that will perform a private genomic test on the customer's sequenced genome. For this use case, in one of our two variants (TXT-only), a company can initialize the device in 5.6 seconds and a customer can perform a test in 34 seconds.

2 Preliminaries

2.1 One-Time Program Background

A one-time program can be conceived of as a non-interactive version of a two party computation: $y = f(a, b)$ where a is Alice's private input, b is Bob's, f is a public function (or program), and y is the output. Alice hands to Bob an implementation of $f_a(\cdot)$ which Bob can evaluate on any input of his choosing: $y_b = f_a(b)$. Once he executes on b, he cannot compute $f_a(\cdot)$ again on a different input. For our practical use-case, we conceive of OTPs with less generality as originally proposed by Goldwasser et al. [17]; essentially we treat them as one-time, non-interactive programs that hide Alice and Bob's private inputs from each other without any strong guarantees on f itself. Note with a general compiler for f (which we have for both flavours of our system), it is easy but

[7] As an example, Intel STK2mv64CC, a Compute Stick that supports both TXT and TPM, was priced at \$499.95 USD on Amazon.com (as of September 2018).

[8] A state-bound cryptographic operation performed by the TPM chip, like encryption.

inefficient to keep f private.[9] Additional background on one-time programs is provided in Appendix A.1.

2.2 Threat Model and Requirements

We informally consider an OTP to be secure if the following properties are achieved: (1) Alice's input a is confidential from Bob; (2) Bob's input b is confidential from Alice, and (3) no more than one b can be executed in $f(a, b)$ per device. We argue the security of our two systems in Section 8 but provide a synopsis here first. Property 3 is enforced through a trusted execution environment, either directly (TXT-only variant in Section 4) or indirectly via a one-time memory device (GC-based TXT in Section 5) as per the Goldwasser et al. construction. Given Property 3, we consider Property 1 to be satisfied if an adversary learns at most negligible information about a when they choose b and observe $\langle \mathsf{OTP}, f(a, b), b\rangle$ as opposed to simply $\langle f(a, b), b\rangle$, where OTP is the entire instantiation of the system, including the TPM-sealed memory and system details (and for the GC-variant: the garbled circuit and keys revealed through specifying b). Property 2 is achieved by being provisioned an offline device that can compute $f_a(b)$ without any interaction with Alice. There is a possibility that the device surreptitiously stores Bob's input and tries to leak it back to Alice. We discuss this systems-level attack in Section 8. In Appendix B, we address a subtle adaptive security attack applicable to the original OTP system.

The seleciton of TEE has to reflect the aforementioned Properties 1 and 3. Property 3 is achieved by stateful (recording the one-time state) and integrity-protected (enforcing one-timeness) execution, which is the fundamental purpose of all today's TEEs. Moreover, both Properties 1 and 3 mandate no information leakage, which can occur through either software or physical side-channels. We choose Intel TXT, primarily because of its *exclusiveness*, which means: TXT occupies the entire system when secure execution is started and no other code can run in parallel. This naturally avoids all software side-channels, an advantage over non-exclusive TEEs. We do consider using non-exclusive TEEs as future exploration when the challange of software side-channels has been overcome, e.g., for Intel SGX, the (recent) continually identified side-channel attacks, such as Foreshadow [9], branch shadowing [40], cache attacks [7], and more; for ARM TrustZone, there have been TruSpy [72], Cachegrab [48], etc. They all point to the situation when trusted and untrusted code run on shared hardware.

The known physical side-channels can also be mitigated in the setting of our OTP, i.e., DMA attacks are impossible if I/O protection is enable (by the chipset), and the cold-boot attack [22] can be avoided if we choose computers with RAM soldered on the motherboard (cannot be removed to be mounted on

[9] Essentially, one would define a very general function we might call Apply that will execute the first input variable on the second: $y = \mathsf{Apply}(f, b) = f(b)$. Since f is now Alice's private input, it is hidden. The implementation of Apply might be a universal circuit where f defines the gates' logic — in this case Apply would leak (an upper-bound on) the circuit size of f but otherwise keep f private.

anohther machine, see Section 8). For a detailed comparison with other TEEs, refer to Appendix A.2).

We strive for a reasonable, real-world threat model where we mitigate attacks introduced by our system but do not necessarily resolve attacks that apply broadly to practical security systems. Specifically, we assume:

- Alice is monetarily driven or at least curious to learn Bob's input, while Bob is similarly curious to learn the algorithm of the circuit and/or re-evaluate it on multiple inputs of his choice.
- We assume Alice produces a device that can be reasonably assured to execute as promised (disclosed source, attestation quotes over an integral channel, and no network capabilities).
- We assume that Alice's circuit (including the function and her input) actually constitutes the promised functionality (e.g., is a legitimate genomic test).
- We assume the sound delivery of the device to Bob. We do not consider devices potentially subverted in transit which applies to all electronics [56].
- Both Alice and Bob have to trust the hardware manufacturer (in our case, Intel and the TPM vendor) for their own purposes. Alice trusts that the circuit can only be evaluated once on a given input from Bob, while Bob trusts that the received circuit is genuine and the output results are trustworthy.
- Bob has only bounded computational power, and may go to some lab effort, such as tapping pins on the motherboard and cloning a hard drive, but not efforts as complicated as imaging a chip [60, 37, 36].
- Components on the motherboard cannot be manipulated easily (e.g., forwarding TPM traffic from a forged chip to a genuine one by desoldering).

2.3 Intel TXT and TPM

Intel Trusted Execution Technology (TXT) is also known as "late launch", for its capability to launch secure execution at any point, occupying the entire system. When the CPU enters the special mode of TXT, all current machine state is discarded/suspended and a fresh secure session is started, hence its exlusiveness, as opposed to sharing hardware with untrusted code.

Components. TXT relies on three mandatory hardware components to function: a) CPU. The instruction set is extended with a few new instructions for the management of TXT execution. b) Chipset. The chipset (on the motherboard) is respondible for enforcing I/O protection such that the specified range of I/O space is only accesible by the protected code in TXT; and c) TPM. Trusted Platform Module [62] is a microchip, serving as the secure storage (termed Secure Element). Its *PCR* (Platform Configuration Register) is volatile storage containing the machine state, in the form of concatenated hash values. There are also multiple PCRs for different purposes. On the TPM, there is also non-volatile storage (termed *NVRAM*), allocated in the unit of *index* of various sizes. Multiple indices can be defined depending on the capacity of a specific TPM model.

Measured launch. A provisioning stage is always involved where the platform is assumed trusted and uncompromised. A piece of code is measured (similar to hashing) and the measurements are stored in certain TPM NVRAM indices as policies. Thereafter (in our case in the normal execution mode with Bob), the program being loaded is measured and compared with the policies stored in TPM. The system may then abort execution if mismatch is detected, or otherwise proceed. This process is enforced by the CPU.

Machine state binding. As run-time secrecy (secret in use) is ensured by measured launch and I/O isolation, we also need secrecy for stored data (secret at rest). Alice's input should not be learned by Bob when the device is shipped to him. From the start of TXT execution, each stage measures the next stage's code and *extends* the hash values as measurement to the PCR (concatenated and hashed with the existing value). This way, the measurements are chained, and at a specific time the PCR value reflects what has been loaded before. The root of this chained trust is the measured launch.

Such chained measurements (in PCRs) can be used to derive the key for data encryption, so that only when a desired software stack is running can the protected data be decrypted. This cryptographic operation performed by the TPM is termed *sealing.* A piece of data sealed under certain PCRs can only be unsealed under the same PCRs, hence bound to a specific machine state. The sealed data (ciphertext) can be stored anywhere depending on its size. It is noteworthy to mention that there exists a distinct equivalent of sealing which, instead of just encryption, stores data in a TPM NVRAM index and binds its access to a set of PCRs. As a resuilt, without the correct machine state, the NVRAM index is completely inaccessible (read/write) and thus replaying the ciphertext is prevented. We term it *PCR-bound NVRAM sealing* in this paper and use it for our OTP prototype implementation.

3 Related Work

In the original one-time program paper by Goldwasser et al. [17], OTM is left as a theoretical device. In the ensuing years, there have been some design suggestions based on quantum mechanisms [8], physically unclonable functions [33], and FPGA circuits [29]. (**a**) Järvinen et al. [29] provide an FPGA-based implementation for GC/OTP, with a GC evaluation of AES, as an example of a complex OTP application. They conclude that although GC/OTP can be realized, their solution should be used only for "truly security-critical applications" due to high deployment and operational costs. They also provide a cryptographic mechanism for protecting against a certain adaptive attack with one-time programs (see Appendix B); it is tailored for situations where the functions output size is larger than the length of a special holdoff string stored at each OTM. (**b**) Kitamura et al. [34] realize OTP without OTM by proposing a distributed protocol, based on secret sharing, between non-colluding entities to realize the 'select one key; delete the other key' functionality. This introduces further interaction and entities. Our approach is in the opposite direction: removing all interaction (other than transfer of the device) from the protocol. (**c**) Prior to OTP being proposed, Gunupudi and Tate [21] proposed count-limited private key usage for realizing non-interactive oblivious transfer using a TPM. Their solution requires changes in the TPM design (due to lack of a TEE). In contrast, we utilize unmodified TPM 1.2. (**d**) In a more generalized setting, ICE [58] and Ariadne [59] consider the state continuity of any stateful program (including N-timeness) in the face of unexpected interruption, and propose mechanisms to ensure both rollback protection and usability (i.e., liveness). We solve the specific problem of one-timeness/N-timeness, focusing more on how to deal with input/output and its implication on performance. We do sacrifice liveness (i.e., we flip the one-timeness flag upon entry and thus the program might run zero time if crashed halfway). We believe their approaches can be applied in conjunction with ours.

4 System 1: TXT-only

Overview. In the first system, we propose to achieve one-timeness by running the protected program in TEE only once (relying on logic integrity) and storing its persistent state (e.g., the one-time indicator) in a way that it is only accessible from within the TEE. To eliminate information leakage from software side-channels, we have chosen Intel TXT for its exclusiveness (i.e., no other software in parallel).[10] We hence name this design *TXT-only*.

To achieve minimal TCB (Trusted Computing Base) and simplicity, we choose native C programming in TXT (as opposed to running an OS/VM). Therefore, for one-time programs that have existing implementation in other languages, per-application adaptation is required (cf. similar porting effort is needed for the GC-based variant in Section 5). For further consideration of the porting effort required, refer to Appendix H. For new programs, this may not introduce extra effort.

Design. We briefly describe the components and workflow of the TXT-only system as follows. A one-time indicator (flag) is sealed into the PCR-bound TPM NVRAM to prevent replay attacks. The indicator is checked and then flipped upon entry of the OTP. Without network connection, the device shipped to the client can no longer leak any of the client's secrets to the vendor. Therefore, only the vendor's secret input has to be protected. We TPM-seal the vendor input on hard drive for better scalability, and there is no need to address replay attacks for vendor input as one-timeness is already enforced with the flag.

The OTP program is loaded by the Intel official project *tboot* [27] and GRUB. It complies with the Multiboot specification [16], and for accessing TPM, we reuse part of the code from tboot, and develop our own functions for commands that are unavailable elsewhere, e.g., reading/writing indices with PCR-bound NVRAM sealing. Since we do not load a whole OS into TXT with tboot, we cannot use OS services for disk I/O access; instead, we implement raw PATA (Parallel ATA, a legacy interface to the hard drive, compatible mode with SATA) logic and directly access disk sectors with DMA (Direct Memory Access). In the *provisioning mode*, the OTP program performs a one-time setup, such as

[10] We consider various TEE options and provide our reasons for choosing Intel TXT to instantiate the TEE in Appendix A.2.

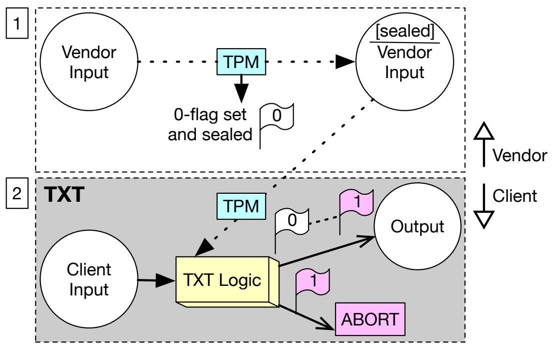

Fig. 1. Our realization of OTPs spans two phases when relying on TXT alone for the entire computation. Alice is active only during phase 1; Bob only during phase 2.

initiating the flag in NVRAM, sealing (overwriting) Alice's secret, etc. Once the *normal execution mode* is entered, the program will refuse to run a second time.

Memory exposure. As an optional feature for certain computers with swappable RAM, we expose the unsealed vendor input in very small chunks during execution. For example, if the vendor input has 100 records, we would unseal one record into RAM each iteration for processing the whole user input. This way, in case of the destructive cold boot attack, the adversary only learns one-hundredth of the vendor's secret, and no more attempts are possible (the indicator is already updated).

4.1 TXT-only provisioning/evaluation

Figure 1 gives an overview of *TXT-only*, illustrating the initial provisioning by Alice and evaluation of the function upon delivery to Bob. Note that what is delivered to Bob is the entire computer in our prototype (laptop or barebone like Intel NUC).

Provisioning at Alice's site. At first, Alice is tasked with setting up the box, which will be delivered to Bob. Alice performs the following: (1) Write the integrity-protected payload/logic in C adapted to the native TXT environment, e.g., static-linking any external libraries and reading input data in small chunks. We may refer to it as the TXT program thereinafter. (2) In the provisioning mode, initialize the flag to 0 and seal.[11] The one-timeness flag is stored with the PCR-bound NVRAM sealing. Instead of depending on a password and regular sealing, this is like stronger access-controlled ciphertext. (3) Seal Alice's input onto the hard drive.

Evaluation at Bob's site. After receiving the computation box from Alice, Bob performs the following: (1) Place the file with Bob's input on the hard drive. (2) Load the TXT program in normal execution mode, which will read in Bob's input and unseal Alice's input to compute on. (3) Receive the evaluation

[11] A flag is more straightforward to implement than a TPM monotonic counter, thanks to the PCR-bound NVRAM sealing, whereas a counter would involve extra steps (such as attesting to the counterAuth password).

result (e.g., from the screen or hard drive). As long as it is Bob's first attempt to run the TXT program, the computation will be permitted and the result will be returned to Bob. Otherwise, the TXT program will abort upon loading in step (2), as shown in Figure 1.

5 System 2: GC-based

As seen in our TXT-only approach to OTP (System 1) the data processing for protection is only applied to Alice's input (with either sealing/unsealing or encryption/decryption), and Bob's input is always exposed in plaintext due to the machine's physical possession by Bob. Intuitively, we may think that it is a good choice when Alice's input is relatively small regardless of Bob's input size. However, there might be other applications where Alice's input is substantially larger and become the performance bottleneck. Is there a construction that complements TXT-only and is less sensitive to Alice's input size? The answer may lie in garbled circuits. During garbled circuit execution, randomly generated strings (or keys) are used to iteratively unlock each gate until arriving at the final output. Alice's input (size) is only "reflected" in the garbled circuit (assumed not trivially invertable [17]), and the key pairs (whose number is determined by Bob's input size, not to do with Alice's) are sealed/encrypted, hence insensitive to Alice's input size. Details of garbled circuits and their use are explained in Appendix A.1.

To adapt garbled circuits for OTP, key generation and key selection steps are separated. As long as we limit key selection to occur a single time, and the unchosen key of each key pair is never revealed, we can prevent running a particular circuit on a different input. To prevent keys from being selected more than once, we need to instantiate a one-time memory (OTM), which reveals the key corresponding to each input bit and effectively destroys (or its equivalent) the unchosen key in the key pair. OTM is left as a theoretical device in the original OTP paper [17]. We realize it using Intel TXT and the TPM. As in System 1, we seal a one-time flag into the PCR-bound TPM NVRAM, and minimize the TXT logic to just handle key selection, in preparation for GC execution. Alice will seal (in advance) key pairs for garbling Bob's inputs. Bob may then boot into TXT to receive the keys corresponding to his input. When Bob reads a key off the device (say for input bit 0), the corresponding key (for input bit 1) is erased.[12] By instantiating an OTM in this manner, we can replace interactive oblivious transfer (OT) and perform the rest of the garbled circuit execution offline, passing key output from trusted selection. By combining TXT and garbled circuits in this way, sealing complexity is now tied to Bob's inputs. We name this alternate construction *GC-based* (System 2).

Performance overhead with TPM sealing. According to our measurement, each TPM sealing/unsealing operation takes about 500ms, and therefore 1 GB of key pairs would need about 1000 hours, which is infeasible. Instead, we generate a random number as an encryption key (MK) at provisioning time and the GC

[12] Unselected keys remain sealed, if never unsealed it serves as cryptographic deletion.

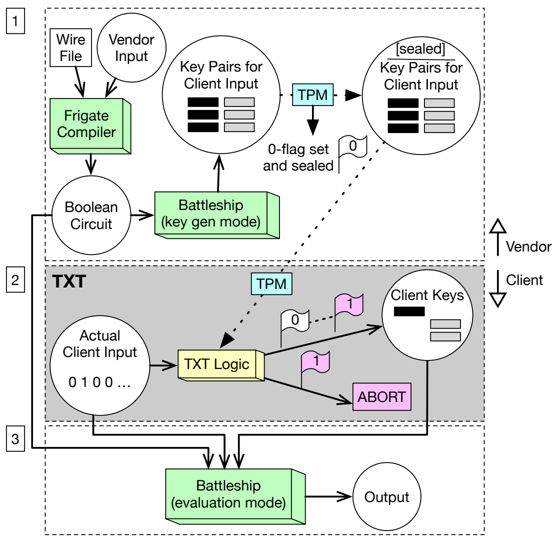

Fig. 2. In our GC-based approach to OTP, Alice generates key pairs and seals them. Bob unseals the keys that correspond to his input and locally evaluates the function. For details about Frigate and Battleship, see Appendix C

key pairs are encrypted with MK. We only seal MK. This way, MK becomes per-deployment, and reprovisioning the system will not make the sealed key pairs reusable due to the change of MK (i.e., the old MK is replaced by the new key). Note that we could also apply the same approach to TXT-only (i.e., encrypting Alice's input with MK and sealing only MK), if needed by the application.

Memory exposure. Similarly to the TXT-only OTP, our GC-based approach can also optionally adapt to address the cold-boot attack. MK becomes a single point of failure if exposed in such memory attacks, i.e., all key pairs can be decrypted and one-timeness is lost. As with TXT-only, for smaller-sized client input, we can seal the key pairs directly and only unseal into RAM in small chunks.

5.1 Implementation

We use the Boolean circuit compiler *Frigate* [44] to implement the garbled circuit components of *GC-based*. The interpreter and execution functionalities of *Frigate* are separately referred to as *Battleship*. For our purposes, we split *Battleship* execution into two standalone phases: a key pair generation phase (`gen`) and a function evaluation phase (`evl`). The motivation behind our choice of *Frigate* and our specific modifications to *Battleship* are detailed in Appendix C.

Our GC-based approach to OTP relies on TXT for trusted key selection and leaves the computation for garbled circuits, as shown in Figure 2. In our setting, Alice represents the vendor and Bob represents the client.

Provisioning at Alice's site. Alice sets up the OTP box by doing the following: (1) Initialize flag to 0 and seal in the TXT program's provisioning mode. (2)

Write and compile, using *Frigate*, the wire program (.wir), together with Alice's input, into the circuit.[13] (3) Load the compiled .mfrig and .ffrig files, vendor's input, and the *Battleship* executable onto the box. (4) Write the TXT program (for key selection) in the same way as in TXT-only. (5) Run *Battleship* in key-generation mode to generate the k_i^0 and k_i^1 key-pairs corresponding to each of the i bits of Bob's input. These are saved to file. (6) Seal the newly generated key pairs onto the hard-drive in provisioning mode of the TXT program. Alice is able to generate the correct number of key pairs, since garbled circuit programs take inputs of a predetermined size, meaning Alice knows the size of Bob's input. Costly sealing of all key pairs could be switched out for sealing of the master key (MK) used to encrypt the key pairs.

Evaluation at Bob's site. Bob, upon receiving the OTP box from Alice, performs the following steps to evaluate the function on his input: (1) Place the file with Bob's input bits on the hard drive. (2) Load the TXT program in normal (non-provisioning) mode for key selection. (3) Receive selected keys corresponding to Bob's input bits; these are output to disk in plaintext. As long as it is Bob's first attempt to select keys, the TXT program will return the keys corresponding to Bob's input. Otherwise, the TXT program will abort upon loading in step (2), as shown in Figure 2. After Bob's inputs have been successfully garbled (or converted into keys) and saved on the disk, Bob can continue with the evaluation properly. TXT is no longer required. (4) Reboot the system into the OS (e.g., Ubuntu). (5) Launch *Battleship* in circuit-evaluation mode. (6) Receive the evaluation result from *Battleship*. When *Battleship* is launched in circuit-evaluation mode, the saved keys corresponding to Bob's input are read in. *Battleship* also takes vendor input (if not compiled into the circuit) before processing the garbled circuit. The Boolean circuit is read in from the .mfrig and .ffrig files produced by *Frigate*. Evaluation is non-interactive and offline. The evaluation result is available only to Bob.

6 Case Study

We apply our proposed systems on a concrete use case based on genomic testing as a protytype. Later in Section 7, we also present another use case of database queries for different input sizes. Further potential use cases are discussed in brief in Appendix E.

Single nucleotide polymorphism (SNP) is a common form of mutation in human DNA. Certain sets of SNPs determine the susceptibility of an individual to specific diseases. Analyzing an individual's set of SNPs may reveal what kind of diseases a person may have. More generally, genomic data can uniquely identify a person, as it not only gives information about a person's association with diseases, but also about the individual's relatives [47]. Indeed, advancements in genomics research have given rise to concerns about individual privacy and

[13] The wire program may be written and compiled on a separate machine from that which will be shipped to Bob. If Alice chooses to use the same machine, the (no longer needed) raw wire code and *Frigate* executable should be removed from the box before provisioning continues.

led to a number of related work in this space. For instance, Canim et al. [10] and Fisch et al. [15] utilize tamper-resistant hardware to analyze/store health records. Other works [66, 5] investigate efficient, privacy-preserving analysis of health data.

While a number of different techniques have been proposed for privacy-preserving genomic testing, ours is the first work to address this using one-time programs grounded in secure hardware. Other than providing one-timeness, the proposed scheme also provides (i) *non-interactivity*, in which the user does not need to interact with the vendor during the protocol, and (ii) *pattern-hiding*, which ensures that the patterns used in vendor's test are kept private from the user. On the other hand, homomorphic encryption-based schemes [3] lack non-interactivity and functional encryption-based schemes [46] lack non-interactivity and pattern-hiding. We did not specifically implement these other techniques and compare our solution with them. However, from the performance results that are reported in the original papers, we can argue that our proposed scheme provides comparable (if not better) efficiency compared to these techniques.

Our aim is to prevent the adversary (the client/Bob), who uses the device for genomic testing, from learning which positions of his genome are checked and how they are checked, specifically for the genomic testing of the breast cancer (BRCA) gene. BRCA1 and BRCA2 are tumor suppressor genes. If certain mutations are observed in these genes, the person will have an increased probability of having breast and/or ovarian cancer [65]. Hence, genomic testing for BRCA1 and BRCA2 mutations is highly indicative of individuals' predisposition to develop breast and/or ovarian cancer.

We aim also to protect the privacy of the vendor (the company/Alice) that provides the genomic testing and prevent the case where the adversary extracts the test, learns how it works, and consequently, tests other people without having to purchase the test. We aim to protect both the locations that are checked on the genome and the magnitude of the risk factor corresponding to that position. Note that client's input is secure, as Bob is provided the device and he does not have to interact with Alice to perform the genomic test.

6.1 Genomic Test

In order to perform our genomic testing, we obtained the SNPs related with BRCA1[14] along with their risk factors from SNPedia [11], an open source wiki site that provides the list of these SNPs. The SNPs that are observed on BRCA1 and their corresponding risk factors for breast cancer are listed in Appendix F.

We obtain genotype files of different people from the openSNP website [19]. The genotype files contain the extracted SNPs from a person's genome. At a high level, for each SNP of the patient that is linked to BRCA1, we add the corresponding risk factor to the overall risk.

The details of our genomic algorithm are shown in Appendix G. If a BRCA1-associated SNP is observed in the patient's SNP file, we check the allele combination and add the corresponding risk factor to the total amount. In order to

[14] Similarly, we can also list the SNPs for BRCA2 and determine the contribution of the observed SNPs to the total risk factor.

prevent a malicious client from discovering which SNPs are checked, we check every line in the patient's SNP file. If an SNP related to breast cancer is not observed at a certain position, we add zero to the risk factor rather than skipping that SNP to prevent the client from inferring checked SNPs using side channels.

Let i denote the reference number of an SNP and s_i^j be the allele combination of SNP i for individual j. Also, S_i and C_i are two vectors keeping all observed allele combinations of SNP i and the corresponding risk factors, respectively. Then, the equation to calculate the total risk factor for individual j can be shown as $RF_j = \sum_i f(s_i^j)$ where

$$f(s_i^j) = \begin{cases} C_i(\ell) & \text{if } s_i^j = S_i(\ell) \text{ for } \ell = 0, 1, \ldots, |S_i| \\ 0 & \text{otherwise} \end{cases}$$

For instance, for the SNP with ID i = rs28897696, $S_i = <AA, AC>$ and $C_i = <7, 6>$. If the allele combination of SNP rs28897696 for individual j corresponds to one of the elements in S_i, we add the corresponding value from C_i to the total risk factor.

6.2 Construction for GC-Based

The garbled circuit version of the genomic test presented in Section 6.1 is written as wire (.wir) code accepted by the *Frigate* garbled circuit compiler. The code follows the algorithm in Appendix G, adjusting overall risk factor upon comparing allele-pairs of matching SNPs and explicitly adding zero when needed.

We choose Bob's input from AncestryDNA files available on the openSNP website [19]. We perform preprocessing on these to obtain a compact representation of the data (specifics are available in Appendix D). Alice's input is hard-coded into the circuit at compile-time, by initializing an `unsigned int` of vendor input size and assigning each bit's value using *Frigate*'s wire operator.

Final input representation. Following the original design of *Battleship*, inputs are accepted as a single string of hex digits (each 4 bits). Each digit is treated separately, and input is parsed byte-by-byte (e.g., 41_{16} is represented as 10000010_2).

We use 7 hex digits (28 unsigned bits) for the SNP reference number and a single hex digit (4 unsigned bits) to represent the allele pair out of 16 possible combinations of A/T/C/G. Alice's input contains 2 more hex digits (8 signed bits) for risk factor, supporting individual risk factor values ranging from -128 to 127. We keep risk factor a signed value, since some genetic mutations lower the risk of disease. Although we did not observe any such mutations pertaining to BRCA1, our representation gives extensibility to tests for other diseases.

Output representation. The program outputs a signed 16-bit value, allowing us to support cumulative risk factor ranging from -32,768 to 32,767.[15]

[15] This can easily be adjusted, but is accompanied by substantial changes in the resulting circuit size. For example, an 11 GB circuit that outputs 16 bits grows to 18 GB by doubling the output size to 32 bits. We conservatively choose 16 bits for demonstration purposes, but the output size may be reduced as appropriate.

6.3 Construction for TXT-only

In TXT-only, the genomic test logic of Section 6.1 is ported in pure C but largely keeps the representation used by the GC program (Section 6.2). Alice's input is in the form of 7 hex digits for the SNP ID, 1 hex digit for the allele pair and 2 digits for the risk factor. Bob's input is 2 digits shorter without the risk factor.

We pay special attention to minimizing exposure of Alice's input in RAM to defend against potential cold-boot attack. We achieve this by processing one record at a time performing all operations on and deleting it before moving on to the next record. We also seal each record (10 bytes) into one sealed chunk (322 bytes), which consumes more space. In each iteration, we unseal one of Alice's records and compare with all of Bob's records. For certain laptops and other computers with RAM soldered on the motherboard, this is optional.

7 Performance evaluation

In this section, we evaluate the two OTP systems' performance/scalability, with varying client and vendor inputs, and try to statistically verify the suitability of the two intuitive designs in different usage scenarios. We perform our evaluation on a machine with a 3.50 GHz i7-4771 CPU, Infineon TPM 1.2, 8 GB RAM, 320 GB primary hard-disk, additional 1 TB hard-disk[16] functioning as a one-time memory (dedicated to storing garbled circuit, and client and vendor input), running Ubuntu 14.04.5 LTS. In one case, we required an alternate testing environment: a server-class machine with a 40 core 2.20 GHz Intel Xeon CPU and 128 GB of RAM.[17] Details specific to the setup of our genomic testing were previously given in Section 6.

We perform experiments to determine the effects of varying either client or vendor input size. Based on the case study, the vendor has 880 bits and the client has 22.4M bits of input, so we use 224 and 880 as the base numbers for our evaluation. We multiply by multiples of 10 to show the effect of order-of-magnitude changes on inputs. We start with 224 for client and 880 for vendor inputs. When varying client input, we fix vendor input at 880 bits. When varying vendor input, we fix client input at 224K bits.

7.1 Benchmarking TXT-only

Varying client input. Table 1 shows the timing results for TXT-only provisioning and execution with fixed vendor input and varying client input size. During provisioning, only the vendor input is sealed, so the provisioning time is constant in all cases. As client input size increases, so does execution time, but moderately. Performance is insensitive to client input size up through the 224K case. Even for the largest (22M) test case, increasing the client input size by two orders of magnitude results only in a slowdown by a factor of 3.5x.

[16] We use a second disk to simulate what is shipped to the client (with all test data consolidated), separate from our primary disk for development.

[17] Another option would have been to upgrade the memory of the initial evaluation machine, but we chose to forgo this, as a test run on the server-class machine revealed that upwards of 60 GB would be required (not supportable by the motherboard).

Client Input (bits)	Prov. (ms)	Exec. (ms)
224	5640.17	9394.58
2K	5640.17	9393.88
22K	5640.17	9388.27
224K	5640.17	9426.56
2M	5640.17	11078.19
22M	5640.17	33427.50

Table 1. TXT-only results with vendor input fixed at 880 bits and varying client input size, averaged over 10 runs. Prov./Exec. refers to the provisioning mode and execution mode respectively.

Vendor Input (bits)	Prov. (ms)	Exec. (ms)
880	5640.17	9426.56
8800	53515.75	92551.43
88000	527026.89	921338.53

Table 2. TXT-only results with client input fixed at 224k bits and varying vendor input size, averaged over 10 runs. Performance of TXT-only is linear and time taken is proportional to vendor input size.

Varying vendor input. Table 2 shows the timing results with fixed client input and varying vendor input size. Although we only tested against three configurations, we see an order-of-magnitude increase in vendor input size is accompanied by an order-of-magnitude increase in both provisioning and execution times.

7.2 Benchmarking GC-based

We use the same experimental setup as used in TXT-only, but with additional time taken by the GC portion. Vendor and client each incur runtime costs from a GC (`gen`/`evl`) and a sealing-based (Prov./Sel.) phase.

Varying vendor input. We are interested in whether *GC-based* is less sensitive to the size of Alice's input than *TXT-only*; see Table 3. Since provisioning (Prov.) involves sealing a constant number of key pairs, and selection (Sel.) is dependent on the unsealing of these key pairs to output one key from each, there is no change. Both *Battleship* `gen` and `evl` mode timing is largely invariant, as well. Whereas System 1 performance was linearly dependent on vendor input size, we observe that *GC-based* (System 2) is indeed not sensitive to vendor input.

Vendor Input (bits)	`gen` (ms)	Prov. (ms)	Sel. (ms)	`evl` (ms)
880	2323.7	4244.03	2508.73	31815.4
8800	3198.7	4244.03	2508.73	32200.4
88000	3286.9	4244.03	2508.73	32000.9

Table 3. GC-based results with client input fixed at 224k bits, varying vendor input size, and encryption of keys by a sealed master key, averaged over 10 runs.

Varying client input. For completeness, we also examine the effects of varying client input size on runtime; see Table 4. Prov. and Sel. stages are both slow as client input size increases, since more key pairs must be sealed/unsealed. `gen` and `evl` times are also affected by an increase in client input bits. Most notably, `evl` demonstrates a near order-of-magnitude slowdown from the 224K case to the 2M case, and the slowdown trend continues into the 22M case (despite using the better-provisioned machine to evaluate the 22M case). We indeed find that TXT-only OTP is complemented by GC-based OTP, where performance is sensitive to client input.

7.3 Analysis

Onto our real-world genomic test (among other padded data sets for the evaluation purpose), Alice's input comprises the 22 SNPs associated with BRCA1, presented in Table 7 of Appendix F. Each SNP entry takes up 40 bits, so Alice's

Client Input (bits)	`gen` (ms)	Prov. (ms)	Sel. (ms)	`evl` (ms)
224	1503.7	843.64	600.55	1350.8
2K	1318.9	906.70	688.62	1631.8
22K	1659.7	991.91	724.24	3643.7
224K	2323.7	4244.03	2508.73	31815.4
2M	16842.8	33934.54	19188.31	305362.8
22M	148387.9*	346606.87	283704.57	3108271*

Table 4. GC-based results with vendor input fixed at 880 bits, varying client input size, and encryption of keys by a sealed master key, averaged over 10 runs. Provisioning- and execution-mode times were measured separately. *s indicate tests run in an alternate environment, due to insufficient memory on our primary testing setup.

OTP Type	Mode	Timing (ms)
TXT-only	Prov.	5640.17
	Exec.	33427.50
GC-based	`gen`	148387.9*
	Prov.	346606.87
	Sel.	283704.57
	`evl`	3108271*

Table 5. Performance for TXT-only and GC-based OTP implementations of the BRCA1 genomic test, averaged over 10 runs. Vendor input is 880 bits. Client input is 22,447,296 bits. *s indicate tests run in an alternate environment, due to insufficient memory on our primary testing setup.

input takes up 880 bits. Bob's input comprises the 701,478 SNPs drawn from his AncestryDNA file, each of which is represented with 32 bits, adding up to a total size of 22,447,296 bits. This genomic test corresponds to our earlier experiment with vendor input size of 880 bits and client input size of 22M bits.

Table 5 puts together the results for both OTP systems. Even at first glance, we see that TXT-only OTP vastly outperforms the GC-based OTP. Provisioning is two orders of magnitude slower in GC-based OTP, and trusted selection itself is an order of magnitude slower than the entire execution mode of TXT-only OTP. `gen` and `evl` further introduce a performance hit to GC-based OTP (again, despite the fact that we evaluated this case on a better-provisioned machine). TXT-only is the superior option for our genomic application.

Small Vendor + Small Client **TXT-only**	Small Vendor + Large Client **TXT-only**
Large Vendor + Small Client **GC-based**	Large Vendor + Large Client **TXT-only**

Table 6. Depending on the input sizes of vendor and client, one system may be preferred to the other. GC-based OTP is favorable when large vendor input is paired with small client input; TXT-only OTP otherwise.

Choosing one OTP. We already saw in Section 7.1 that TXT-only OTP is less sensitive to client input, whereas we saw in Section 7.2 that GC-based OTP is less sensitive to vendor input. We illustrate the four cases in Table 6 in four quadrants.

In this specific use-case of genomic testing, we are in the upper-right quadrant and thus the TXT-only OTP dominates. However, other use cases, like the database querying examples in Appendix E, occupy the lower-left quadrant, in which case GC-based will outperform the TXT-only OTP. What should we do if both inputs are of similar size (i.e., equally "small" or "large")? A safe bet is to stick with the TXT-only OTP. Even though GC technology continues to improve, garbled circuits will always be less efficient than running the code natively.

7.4 Another use case: database queries.

To give an example where the vendor input can be significantly large, we may consider another potential and feasible application of our proposed OTP designs, where GC-based can outperform TXT-only. It is also in a medical setting where the protocol is between two parties, namely a company that owns a database consisting of patient data and a research center that wants to utilize patient data. The patient data held at the company contains both phenotypical and genotypical properties. The research center wants to perform a test to determine the relationship of a certain mutation (e.g., a SNP) with a given phenotype. There may be three approaches for this scenario:

1. **Private information retrieval [12]:** PIR allows a user to retrieve data from a database without revealing what is retrieved. Moreover, the user also does not learn about the rest of the data in the database (i.e., symmetric PIR [50]). However, it does not let the user compute over the database (such as calculating the relationship of a certain genetic variant with a phenotype among the people in the database).
2. **Database is public, query is private:** The company can keep its database public and the research center can query the database as much as it wants. However, with this approach the privacy of the database is not preserved. Moreover, there is no limit to the queries that the research center does.
   As an alternative to this, database may be kept encrypted and the research center can run its queries on the encrypted database (e.g., homomorphic encryption). The result of the query would then be decrypted by the data owner at the end of the computation [32]. However, this scheme introduces high computational overhead.
3. **Database is not public, query is exposed:** In this approach, the company keeps its database secret and the research center sends the query to the company. This time the query of the research center is revealed to the company and the privacy of the research center is compromised.

In the case of GC-based, the company stores its database into the device (in the form of garbled circuit) and the research center purchases the device to run its query (in TXT) on it. This system enables both parties' privacy. The device does not leak any information about the database and also the company does not learn about the query of the research center, as the research center purchases the device and gives the query as an input to it. In order to determine the relationship of a certain mutation to a phenotype, chi-squared test can be used to determine the p-value, that helps the research center to determine whether a mutation has a significant relation to a phenotype. We leave this to future work.

8 Security analysis

a) Replay attacks. The adversary may try to trick the OTP into executing multiple times by replaying a previous state, even without compromising the TEE, or the one-time logic therein. The secrets (e.g., MK) only have per-deployment freshness (fixed at Alice's site). Nevertheless, in our implementation, the TPM NVRAM indices where the one-timeness flag and MK are stored are configured with PCR-bound protection, i.e., outside the correct environment, they are even inaccessible for read/write, let alone to replay.

b) Memory side-channel attacks. Despite the hardware-aided protection from TEE, sensitive plaintext data must be exposed at certain points. For instance, MK is needed for encrypting/decrypting key pairs, and the key pairs when being selected must also be in plaintext. Software memory attacks [9, 41, 35] do not apply to our OTP systems, as the selected TEE (TXT) is exclusive. In our design, the code running in TEE does not even involve an OS, driver, hypervisor, or any software run-time. There are generally two categories of physical memory attacks: non-destructive ones that can be repeated (e.g., DMA attacks [52]); and the destructive (only one attempt) physical cold-boot attack [22]. All I/O access (especially DMA) is disabled for the TEE-protected regions and thus DMA attacks no longer pose a threat.

The effective cold-boot attack requires that the RAM modules are swappable and plaintext content is in RAM. For certain laptops or barebone computers [30], their RAM is soldered on the motherboard and completely unmountable (and thus immune). To ensure warm-boot attacks [64] (e.g., reading RAM content on the same computer by rebooting it with a USB stick) are also prevented, we can set the Memory Overwrite Request (MOR) bit to signal the UEFI/BIOS to wipe RAM on the next reboot before loading any system (cf. the official TCG mitigation [61]). We do take into account the regular desktops/laptops vulnerable to the cold-boot attack: For small-sized secrets like MK, existing solutions [45, 20, 63, 53] can be used, where CPU/GPU registers or cache memory are used to store secrets. For larger secrets, like the key pairs/vendor input, we perform block-wise processing so that at any time during the execution, only a very small fraction is exposed. Also, as cold-boot attack is destructive, the adversary will not learn enough to reveal the algorithm or reuse the key pairs. At least, the vendor can always choose computers with soldered-down RAM.

c) Attack cost. Bob may try to infer the protected function and vendor inputs by trying different inputs in multiple instances. This attack may incur a high cost as Bob will need to order the OTP from Alice several times. This is a limitation of any offline OTP solution, which can only guarantee one query per box.

d) Cryptographic attacks. The security of one-time programs (and garbled circuits) is proven in the original paper [17] (updated after caveat [6]), so we do not repeat the proofs here.

e) Clonability. Silicon attacks on TPM can reveal secrets (including the Endorsement Key), but chip imaging/decapping requires high-tech equipment. Thus, cloning a TPM or extracting an original TPM's identity/data to populate a virtual TPM (vTPM) is considered unfeasible. Sealing achieves platform-state-binding without attestation, so non-genuine environments (including vTPM) will fail to unseal. Refer to Appendix I for TPM relay attack and SMM attacks. Furthermore, there has been a recent software attack [23] that resets and forges PCR values during S3 processing exploiting a TPM 2.0 flaw (SRTM) and a software bug in tboot (DRTM). They (allegedly patched) do not pose a threat to our OTP design, as neither SRTM nor any OS software (e.g., Linux) is involved, not to mention our OTP does not support/involve any power management.

f) Input credibility/correctness for genomic tests. Genomic tests should be run with the user's consent and an attacker shouldn't be able to run tests with fake genomes to infer the test. We can use digital signatures to provide credibility, and use biometric attributes to ensure ownership. Another related concern is *inference attacks:* Genetic disorders may be highly correlated with each other; e.g., the SNP with ID rs429358 has influence on the risk of having both Alzheimer's disease and heart disease [55]. Thus the result of a single genomic test can still give information about other diseases. Moreover, circuits should be designed with the same circuit depth to prevent Bob from inferring Alice's input. Additionally, unless a (projective) garbling scheme that realizes an OTP is designed to provide adaptive privacy, it opens up room for *adaptive attacks* [6].[18] Solutions include: allowing the adversary to decrypt the circuit but not learn the output of the circuit until all keys have been chosen [17]; encrypting the circuit using either a one-time-pad or random-oracle-based encryption and revealing the decryption key together with the garbled input in the online phase [6]; and placing a "holdoff" gate into each output wire that cannot be evaluated until all keys are learned [29].

9 Concluding Remarks

Until now, one-time programs have been theoretical or required highly customized/expensive hardware. We shift away from crypto-intensive approaches to the emerging but time-tested trusted computing technologies, for a practical and affordable realization of OTPs. With our proposed techniques, which we will release publicly, anyone can build a one-time program today with off-the-shelf devices that will execute quickly at a moderate cost. The cost of our proposed hardware-based solution for a single genomic test can be further diluted by extension to support multiple tests and multiple clients on a single device (which our current construction already does). The general methodology we provide can be adapted to other trusted execution environments to satisfy various application scenarios and optimize the performance/suitability for existing applications.

References

1. AMD, Inc.: Virtualization Solutions. `http://www.amd.com/en-us/solutions/servers/virtualization` (2017)
2. Apple.com: iOS security guide (2018), white Paper. Available at `https://www.apple.com/business/docs/iOS_Security_Guide.pdf`
3. Ayday, E., Raisaro, J.L., Laren, M., Jack, P., Fellay, J., Hubaux, J.P.: Privacy-preserving computation of disease risk by using genomic, clinical, and environmental data. In: Proceedings of USENIX Security Workshop on Health Information Technologies (HealthTech'13). No. EPFL-CONF-187118 (2013)

[18] For certain classes of circuits, Jafargholi and Wichs [28] claim that garbled circuits are adaptively secure without further modification, with security loss tied to pebble complexity of the circuit.

4. Azema, J., Fayad, G.: M-Shield mobile security technology: making wireless secure. Tech. rep., Texas Instruments (2008)
5. Baldi, P., Baronio, R., De Cristofaro, E., Gasti, P., Tsudik, G.: Countering gattaca: efficient and secure testing of fully-sequenced human genomes. In: Proceedings of the 18th ACM conference on Computer and communications security. pp. 691–702. ACM (2011)
6. Bellare, M., Hoang, V.T., Rogaway, P.: Adaptively secure garbling with applications to one-time programs and secure outsourcing. In: ASIACRYPT (2012)
7. Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., Sadeghi, A.R.: Software grand exposure: SGX cache attacks are practical. In: 11th USENIX Workshop on Offensive Technologies (WOOT 17). Vancouver, BC (2017)
8. Broadbent, A., Gutoski, G., Stebila, D.: Quantum one-time programs. In: CRYPTO. pp. 344–360 (2013)
9. Bulck, J.V., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In: USENIX Security Symposium. pp. 991–1008. Baltimore, MD, USA (2018)
10. Canim, M., Kantarcioglu, M., Malin, B.: Secure management of biomedical data with cryptographic hardware. IEEE Transactions on Information Technology in Biomedicine **16**(1), 166–175 (2012)
11. Cariaso, M., Lennon, G.: SNPedia: a wiki supporting personal genome annotation, interpretation and analysis (2010), `http://www.SNPedia.com`
12. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on. pp. 41–50. IEEE (1995)
13. Ermolov, M., Goryachy, M.: How to hack a turned-off computer or running unsigned code in intel management engine. Tech. rep., Black Hat Europe (2017)
14. Fink, R.A., Sherman, A.T., Mitchell, A.O., Challener, D.C.: Catching the cuckoo: Verifying tpm proximity using a quote timing side-channel. In: McCune, J.M., Balacheff, B., Perrig, A., Sadeghi, A.R., Sasse, A., Beres, Y. (eds.) Trust and Trustworthy Computing. pp. 294–301. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
15. Fisch, B.A., Vinayagamurthy, D., Boneh, D., Gorbunov, S.: Iron: Functional encryption using Intel SGX. Tech. rep., IACR eprint (2016)
16. Gnu.org: The multiboot specification (2009), `http://www.gnu.org/software/grub/manual/multiboot/multiboot.html`
17. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-Time Programs. In: CRYPTO. pp. 39–56 (2008)
18. Greene, J.: Intel® Trusted Execution Technology. Tech. rep. (2012)
19. Greshake, B., Bayer, P.E., Rausch, H., Reda, J.: Opensnp–a crowdsourced web resource for personal genomics. PLoS One **9**(3), 1–9 (2014)
20. Guan, L., Lin, J., Luo, B., Jing, J.: Copker: Computing with private keys without RAM. In: NDSS. San Diego, CA, USA (Feb 2014)
21. Gunupudi, V., Tate, S.R.: Generalized non-interactive oblivious transfer using count-limited objects with applications to secure mobile agents. In: Financial Cryptography and Data Security. pp. 98–112. FC'08 (2008)
22. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: Cold boot attacks on encryption keys. In: USENIX Security Symposium. San Jose, CA, USA (2008)

23. Han, S., Shin, W., Park, J.H., Kim, H.: A bad dream: Subverting trusted platform module while you are sleeping. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 1229–1246. Baltimore, MD, USA (2018)
24. Hazay, C., Lindell, Y.: Efficient Secure Two-Party Protocols. Springer (2010)
25. Holzer, A., Franz, M., Katzenbeisser, S., Veith, H.: Secure Two-party Computations in ANSI C. In: CCS. pp. 772–783 (2012)
26. Intel Corporation: Intel Software Guard Extensions (Intel SGX). `https://software.intel.com/en-us/sgx` (2016)
27. Intel Corporation: Trusted boot (tboot) (2017), version: 1.8.0. `http://tboot.sourceforge.net/`
28. Jafargholi, Z., Wichs, D.: Adaptive Security of Yao's Garbled Circuits. In: TCC. pp. 433–458 (2016)
29. Järvinen, K., Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs. In: CHES. pp. 383–397. CHES'10 (2010)
30. Jefferies, C.P.: How to identify user-upgradeable notebooks (June 2017), web article. Available at `http://www.notebookreview.com/feature/identify-user-upgradeable-notebooks/`
31. Johnson, S.: Intel® SGX and Side-Channels. `https://software.intel.com/en-us/articles/intel-sgx-and-side-channels` (2017)
32. Kantarcioglu, M., Jiang, W., Liu, Y., Malin, B.: A cryptographic approach to securely share and query genomic sequences. IEEE Transactions on information technology in biomedicine **12**(5), 606–617 (2008)
33. Kirkpatrick, M.S., Kerr, S., Bertino, E.: PUF ROKs: A hardware approach to read-once keys. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. pp. 155–164. AsiaCCS'11, Hong Kong, China (2011)
34. Kitamura, T., Shinagawa, K., Nishide, T., Okamoto, E.: One-time Programs with Cloud Storage and Its Application to Electronic Money. In: APKC (2017)
35. Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Yarom, Y.: Spectre attacks: Exploiting speculative execution. CoRR (2018)
36. Kollenda, B., Koppe, P., Fyrbiak, M., Kison, C., Paar, C., Holz, T.: An exploratory analysis of microcode as a building block for system defenses. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 1649–1666 (2018)
37. Koppe, P., Kollenda, B., Fyrbiak, M., Kison, C., Gawlik, R., Paar, C., Holz, T.: Reverse engineering x86 processor microcode. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 1163–1180. Vancouver, BC (2017)
38. Kreuter, B., Shelat, A., Mood, B., Butler, K.: PCF: A Portable Circuit Format for Scalable Two-Party Secure Computation. In: USENIX Security Symposium. pp. 321–336 (2013)
39. Kreuter, B., Shelat, A., Shen, C.: Billion-Gate Secure Computation with Malicious Adversaries. In: USENIX Security Symposium. pp. 285–300 (2012)
40. Lee, S., Shih, M.W., Gera, P., Kim, T., Kim, H., Peinado, M.: Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 557–574. Vancouver, BC (2017)
41. Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., Hamburg, M.: Meltdown. CoRR (2018)
42. Matetic, S., Kostiainen, K., Dhar, A., Sommer, D., Ahmed, M., Gervais, A., Juels, A., Capkun, S.: Rote: Rollback protection for trusted execution. Tech. rep., ETH Zurich (2017)

43. McCune, J.M.: Reducing the trusted computing base for applications on commodity systems. Ph.D. thesis, Carnegie Mellon University (2009)
44. Mood, B., Gupta, D., Carter, H., Butler, K., Traynor, P.: Frigate: A Validated, Extensible, and Efficient Compiler and Interpreter for Secure Computation. In: Euro-SP (2016)
45. Müller, T., Freiling, F.C., Dewald, A.: TRESOR runs encryption securely outside RAM. In: USENIX Security Symposium. San Francisco, CA, USA (Aug 2011)
46. Naveed, M., Agrawal, S., Prabhakaran, M., Wang, X., Ayday, E., Hubaux, J.P., Gunter, C.: Controlled functional encryption. In: CCS. pp. 1280–1291. ACM (2014)
47. Naveed, M., Ayday, E., Clayton, E.W., Fellay, J., Gunter, C.A., Hubaux, J.P., Malin, B., Wang, X., et al.: Privacy and security in the genomic era. In: CCS (2014)
48. nccgroup: Cachegrab (December 2017), available at `https://github.com/nccgroup/cachegrab`
49. Ngabonziza, B., Martin, D., Bailey, A., Cho, H., Martin, S.: Trustzone explained: Architectural features and use cases. In: Collaboration and Internet Computing (CIC), 2016 IEEE 2nd International Conference on. pp. 445–451. IEEE (2016)
50. Saint-Jean, F.: Java Implementation of a Single-Database Computationally Symmetric Private Information Retrieval (cSPIR) Protocol. Tech. rep., Yale University Department of Computer Science (2005)
51. Schwarz, M., Weiser, S., Gruss, D., Maurice, C., Mangard, S.: Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA (2017)
52. Sevinsky, R.: Funderbolt: Adventures in Thunderbolt DMA attacks, black Hat USA, 2013
53. Simmons, P.: Security through Amnesia: A software-based solution to the cold boot attack on disk encryption. In: ACSAC (2011)
54. SNPedia: Magnitude. `https://www.snpedia.com/index.php/Magnitude` (2014)
55. SNPedia: rs429358. `https://www.snpedia.com/index.php/Rs429358` (2017)
56. Sottek, T.: NSA reportedly intercepting laptops purchased online to install spy malware (December 2013), web article. Available at `https://www.theverge.com/2013/12/29/5253226/nsa-cia-fbi-laptop-usb-plant-spy`
57. Spivey, H.C.C.D.M.K.S.C.N.A., Smith, R.: Essentials of Genetics. NPG Education (2009)
58. Strackx, R., Jacobs, B., Piessens, F.: Ice: A passive, high-speed, state-continuity scheme. In: Proceedings of the 30th Annual Computer Security Applications Conference. pp. 106–115. ACSAC'14, New Orleans, Louisiana, USA (2014)
59. Strackx, R., Piessens, F.: Ariadne: A minimal approach to state continuity. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 875–892. Austin, TX (2016)
60. Tarnovsky, C.: Attacking tpm part 2: A look at the ST19WP18 tpm device (July 2012), dEFCON presentation. Available at `https://www.defcon.org/html/links/dc-archives/dc-20-archive.html`
61. Trusted Computing Group: TCG Platform Reset Attack Mitigation Specification (May 2008)
62. Trusted Computing Group: Trusted Platform Module Main Specification, version 1.2, revision 116. `https://trustedcomputinggroup.org/tpm-main-specification/` (2011)
63. Vasiliadis, G., Athanasopoulos, E., Polychronakis, M., Ioannidis, S.: PixelVault: Using GPUs for securing cryptographic operations. In: CCS. Scottsdale, AZ, USA (Nov 2014)

64. Vidas, T.: Volatile memory acquisition via warm boot memory survivability. In: 43rd Hawaii International Conference on System Sciences. pp. 1–6 (Jan 2010)
65. Walsh, T., Lee, M.K., Casadei, S., Thornton, A.M., Stray, S.M., Pennil, C., Nord, A.S., Mandell, J.B., Swisher, E.M., King, M.C.: Detection of inherited mutations for breast and ovarian cancer using genomic capture and massively parallel sequencing. Proceedings of the National Academy of Sciences **107**(28), 12629–12633 (2010)
66. Wang, X.S., Huang, Y., Zhao, Y., Tang, H., Wang, X., Bu, D.: Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In: CCS. pp. 492–503. ACM (2015)
67. Wiklander, J.: Secure storage in OP-TEE, available at `https://github.com/OP-TEE/optee_os/blob/master/documentation/secure_storage.md`
68. Wojtczuk, R., Rutkowska, J.: Attacking Intel trusted execution technology (Feb 2009), black Hat DC
69. Wojtczuk, R., Rutkowska, J., Tereshkin, A.: Another way to circumvent Intel trusted execution technology (Dec 2009), technical Report. `http://invisiblethingslab.com/resources/misc09/Another\%20TXT\%20Attack.pdf`
70. Xu, Y., Cui, W., Peinado, M.: Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In: IEEE Symposium on Security and Privacy (2015)
71. Yao, A.C.: Protocols for secure computations. In: FOCS (1982)
72. Zhang, N., Sun, K., Shands, D., Lou, W., Hou, Y.T.: Truspy: Cache side-channel information leakage from the secure world on ARM devices. IACR Cryptology ePrint Archive **2016**, 980 (2016)

The appendices are organized as follows:

- Appendix A provides additional background helpful for understanding on one-time programs, garbled circuits, and one-time memories;
- Appendix B addresses an adaptive security attack on OTP systems;
- Appendix C describes in detail modifications we make to *Battleship*;
- Appendix D presents preprocessing steps for our case study application;
- Appendix E provides additional one-time program use cases;
- Appendix F lists the SNPs associated with BRCA1;
- Appendix G gives our genomic algorithm;
- Appendix H comments on porting efforts required for OTP; and
- Appendix I discusses SMM and TPM relay attacks.

A Additional Background

A.1 One Time Programs

A one-time program (OTP), as introduced by Goldwasser et al. [17], is an implementation of a deterministic function which is provided by Alice to Bob. We describe it here with less generality than it was presented originally (see Section 2.2 for a reconciliation of both approaches). Consider the implementation as containing the function itself (unprotected) and Alice's input to the function (cryptographically protected). Bob can choose a single input and evaluate the function (with Alice's input) on it. With the output, he may be able to infer something about Alice's input (depending on the exact function), but he cannot infer anything about her input beyond this. Since Bob is operating the device autonomously from Alice, his input is unconditionally private from Alice. The core requirement of OTP is that while Bob is able to receive the evaluation on a single input of his choosing, he is unable to obtain an evaluation of any other input. The mechanism to enforce this is the topic of this paper.

The term 'one-time program' is a slight misnomer. One might equate it with a form of copy protection or digital rights management (DRM). It is worth illustrating the difference with a simple example. Consider a DRM scenario: Alice providing Bob with a media player (the function) and a movie (Alice's input). The movie will play if Bob inputs a correct access code. In this case, the stream of the movie is the output of the function; once Bob learns the output, he can replay it as many times as he wants. Therefore, this is not a valid application of OTP. Instead, consider the following: Alice provides Bob with a game of Go (the function) programmed with the latest in artificial intelligence (Alice's input). Bob's moves are his input to the function. He can 'replay' the game with the exact same moves (resulting in the exact same game and outcome) as many times as he wants (so it is not strictly 'one-time'), however as soon as he deviates with a different move, the program will not continue playing. In this sense, he can only play 'once,' limited to a single sequence of moves.

OTPs can be realized in a straightforward way with trusted execution. From here, we will describe the alternative approach [17] of realizing one-time programs via a simpler primitive called one-time memory (OTM), and composing OTM with garbled circuits.

Garbled Circuits. Garbled circuits (GC) were first proposed by Yao [71] as a technique for achieving secure multiparty computation by at least two parties, a *generator* (Alice) and an *evaluator* (Bob). A program is first converted into its Boolean circuit representation. For each of the i wires in the circuit, Alice chooses encryption keys k_i^0 and k_i^1. Each gate of the circuit takes on the form of a truth table, and entries of the truth table are permuted to further conceal whether any particular entry holds a 0- or 1-value. The keys received on each input wire unlocks a single entry of the truth table, itself a key that is released on the output wire and fed into the next gate. Bob receives the garbled circuit from Alice, together with Alice's garbled inputs. Bob garbles its own inputs through oblivious transfer (OT) with Alice. During evaluation, an output key is iteratively unlocked, or decrypted, from each of the garbled gates until arriving at the final output, which is revealed to all participants.

One-Time Memory. In summary, one-time programs extend garbled circuits where the oblivious transfer phase is replaced with a special purpose physical device called one-time memory (OTM). The protocol proceeds as in garbled circuits with Bob given the circuit, encoded with Alice's input under encryption. Instead of interacting with Alice to learn the keys that correspond to his input, Alice provisions a device with all keys on it. However when Bob reads a key off the device (say for input bit 0) the corresponding key (for input bit 1) is erased. The end result is the same as oblivious transfer: Bob receives exactly one key for each input bit while not learning the other key, whereas Alice does not learn which keys Bob selected. The main difference is that key-selection by OTM is non-interactive, meaning Alice can be completely offline.

A.2 Trusted execution environments

Based on the desired system properties defined in Section 2.2, we now discuss the requirements a candidate TEE should satisfy.

- **R1**: Isolated execution with integrity. This corresponds to Property 3, so that the logic is properly enforced and no additional runs are allowed. Most TEEs have this fundamental feature.
- **R2**: Non-volatile secure storage (formally termed Secure Element). Particular to OTP, intuitively a non-volatile flag is needed to record if the program has been run (or how many times). This is required for all stateful programs.
- **R3**: Sealing (machine state binding). For Property 1, the non-run-time secrecy of a needs the capability to bind it to the exact desired machine state, and only under this state can it be retrieved. Such capability is usually called sealing in most TEEs.
- **R4**: No software side-channels. To ensure run-time secrecy, there should be no way for other code (if any) on the same device to learn the secret (or other protected data). Exclusive TEEs naturally satsfy this requirement.
- **R5**: No physical side-channels. There should be no physical side-channels, which specifically refers to either (DMA-related) memory attacks and the cold-boot attack.

Even if not all the requirements (**R1 - R5**) can be satisfied by a particular TEE, we would like to see which is best-positioned to realize OTP and what additional steps can be taken to compensate for those that are missing.

Trusted computing (where TEEs belong) already has a history of more than a decade (cf. an earlier endeavor of Texas Instrument M-Shield [4] on OMAP). TEEs are usually architecture-shipped, with a primary focus on securing processor execution. They can be categorized as one of the following:

1. Exclusive. Exemplified by Intel TXT, this type of TEE suspends all other operations on the processor and owns all resources before it exits. The advantage is less attack vectors exposed.
2. Concurrent. Represented by Intel SGX and ARM TrustZone, this type creates secure enclaves or worlds that exist alongside other processes. There might be multiple instances at the same time. These are more suitable for application-level logic.

We now present a few of the typical TEE options in the context of OTP, and discuss their suitability for matching each of our stated requirements. All TEEs satisfy **R1**, without need for explicit deliberation.

Intel TXT [18] and AMD SVM [1]. TXT and SVM are simply counterparts on their respective vendor's platform, with nearly the same properties (slight differences). They are exclusive by nature and rely on a security chip called TPM (Trusted Platform Module), corresponding to **R2**. When the secure session is started anytime, TXT/SVM measures the loaded binaries and stores the results in TPM. Two primitives are important: 1) Measured launch. TXT/SVM can compare the measurements with the "good" values in TPM and aborts execution if mismatch occurs. 2) Sealing (platform binding, satisfying **R3**). Sealed data can only be accessed in the intact, genuine program and correct platform. Their exclusiveness naturally meets **R4**, as no other code can be run simultaneously. As desktop processors, detachable RAM modules are inevitable, so the cold-boot attack fails **R5**. We will discuss a workaround in Section 4.

ARM TrustZone [49]. TrustZone introduces the notion of secure world and normal world. The secure world coexists with the normal world, with everything (including I/O) separated. The two can communicate through a special monitor. This leaves it questionable for **R4**, as there might be potential side-channel attacks from code running in the normal world. Since it is coupled with the ARM architecture, we can use it on mobile platforms or a dedicated device other than a desktop. This intrinsically satisfies **R5** as it should be difficult (if not impossible) to physically extract RAM secrets, e.g., by probing or detaching memory modules. TrustZone also supports sealing satisfying **R3**. An obvious advantage of TrustZone is its secure peripheral communication (enabled by the AMBA3 AXI to APB Bridge). For example, if a small region of the screen is allocated to the secure world, user input there cannot be intercepted by the normal world OS. However, in our OTP, we have no need to involve a regular full-blown OS. Moreover, one of its disadvantages is that the essential secure element (where secrets are stored, like TPM) is not standardized and always vendor-specific [67], thus failing **R2**. This means for any OTP we develop, we have to collaborate closely with the device manufacturer, whereas for TXT, we can buy COTS devices. Nevertheless, if such collaboration became possible for a specific organization, making use of TrustZone on mobile platforms can significantly lower the cost (from approximately \$500 to a few dollars) per device.

Intel SGX [26]. More recent than TXT, SGX (Software Guard Extensions) can also be utilized to achieve one-timeness. Intel SGX provides finer-grained isolated environment (measurement-based like TXT) where individual secured apps (called *enclaves*) coexist with the untrusted operating system (thus failing **R4**). The integrity of the program logic (e.g., refusing to run a second time) is guaranteed by the measurement of enclaves before loading. However, what was missing has been a secure persistent storage for the flag (to ensure one-timeness) and Alice's input (SGX did not use TPM in the first place); without secure storage, Bob can simply make a copy of both before execution/evaluation, hence defeating one-timeness. To bring back freshness with SGX-sealed data, Intel recently added support for non-volatile on-chip monotonic counters (similar to TPM, stored in the SPI flash chip), see ROTE [42]. Therefore, SGX-sealing the flag and key pairs with replay attack resistance is feasible now (**R3**). We consider its **R2** to be partial, as there is not dedicated general-purpose secure storage like TPM. With respect to **R5**, SGX enclaves' memory is always encrypted outside the CPU, thus immune to the cold-boot attack.

There have also been other TEEs around that are not discussed here, for the reason that either they are less-used or obsoleted (e.g., M-Shield [4]) or no sufficient public information is available to support development (e.g., Apple Secure Enclave Co-processor [2]). We decide to implement our engineering prototype with Intel TXT with the following considerations (compared with SGX). Note that since TrustZone requires vendor collaboration, we skip it for now.

1. **Fewer known flaws.** TXT has been time-tested and known flaws are already stable public information (see Section 8). For SGX, there have been multiple reports regarding various side-channel attacks mounted by malicious/compromised OS or even peer apps [70, 51]. What is worse, Intel admits it as a known flaw that will remain, leaving the closing of side-channels as a responsibility of enclave developers [31]. In other words, side-channels are explicitly out of the thread model of SGX. Such a flaw allows potential multiple or even unlimited number of executions of the protected program, which Bob is motivated to do. On the other hand, although TXT used to have a few system/hardware-level flaws [68, 69] (as no other software can coexist), there are no recent such reports, and previous ones have been patched or not-applicable any more with newer CPU versions. Note that certain attacks based on SMM (System Management Mode) have also been targeting TXT, but does not pose as much threat here, as explained in Section I.
2. **Meltdown [41]/Spectre [35]/Foreshadow [9].** The lately identified flaws in modern processors make side-channel attacks potentially ubiquitous, due to the fact that out-of-order execution is a common feature of modern architectures. What make it worse is the Foreshadow attack specifically targeting

SGX (L1 Terminal Fault). Even after a microcode patch, the trustworthiness is still gone as there is no guarantee that the unique per-CPU key was not exposed before the patch. All such intrinsic side-channel attacks stem from multitasking (co-existing programs). Therefore,the *exclusive* trusted environment (where no other OS/entities/processes exist) is more preferable in achieving one-timeness, which is the case for TXT.

3. **Dedicated environment.** SGX is positioned differently than TXT and does not replace it, in the sense that the former allows multiple user-space instances for cloud applications, whose attestation requires contacting Intel's IAS server each time. In contrast, TXT is a dedicated environment, with reduced attack vectors, that also allows local attestation.

B Adaptive Attacks on OTPs

A subtle security issue arises when comparing the views in the GC-based variant: $\langle \mathsf{OTP}, f(a,b), b\rangle$ and $\langle f(a,b), b\rangle$. In the former case, Bob specifies b bit-by-bit and starts to learn information about $f(a,b)$ before choosing all bits of b. Bob can thus adaptively choose the next bits of b based on his observations about the path through the circuit, and the number of possible paths is generally exponential in the number of gates. Thus, in a formal simulation-based proof, the simulator cannot add a convincing OTP to the second view without knowing all of b a priori.[19] Goldwasser et al. add an all-or-nothing transformation to the output of $f(a,b)$: the output is masked by random bits that are only fully revealed once all of b is selected.[20] In the TXT-only variant, the output is provided at the end as enforced by the execution of the system.

C The Frigate GC compiler

Frigate. Frigate [44] is a modern Boolean circuit compiler that outperforms several other garbled-circuit compilers (e.g., PCF [38], Kreuter et al. [39], CBMC [25]) by orders of magnitude. *Frigate* is also extensively validated and found to produce correct and functioning circuits where other compilers fail [44]. For these reasons, we decide to use *Frigate* for implementing the garbled circuit components of our GC-based OTP.[21]

Battleship. Battleship, developed by the same group, separates out the interpreter and execution functionalities of *Frigate*. Battleship reads in and interprets

[19] This problem does not arise in garbled circuits since the oblivious transfer is completed prior to providing the circuit.

[20] This fixes the issue because the simulator can equivocate on the final masking value to program the random value, sitting wherever the circuit ends up, with the correct output value for the now-known input.

[21] Although we choose to go with *Frigate*, it is possible to instantiate our OTP system with other garbled circuit compilers.

the circuit file produced by *Frigate*. *Battleship* is originally designed to be run interactively by at least two parties, a generator and an evaluator. The generator is able to independently garble its own inputs, whereas the evaluator depends on OT to garble its inputs. At each gate of the garbled circuit, a single value is decrypted from the associated truth table containing encrypted entries. Garbled gates are iteratively decrypted until arriving at the final output, which is released to either party. Output need not be the same for both parties.

To make *Battleship* support one-time programs, we do the following:

- Split execution in *Battleship* into two standalone phases. In the first phase, a fresh set of random keys (0- and 1-keys for encoding each bit of the client's input) is generated and written out to a file. The key pairs contained in this file will be used during TXT provisioning, after which the file is discarded. The second phase reads in another file containing the subset of keys chosen (during trusted selection) according to the client's input bits and performs evaluation of the circuit. *Battleship* did not originally require these file operations since inputs were garbled and immediately usable without needing to interrupt the system, while we rely on Intel TXT.
- Remove the oblivious-transfer step. In our setting, vendor and client do not perform interactive computation in real time. Instead, the client receives the garbled representation of its input during the trusted selection process inside Intel TXT. The client's input chosen in this way is not exposed to the vendor, who no longer has access to the system after sending it to the client.
- Remove dependency on the full set of 0- and 1- keys in the second phase. In the original *Battleship* design, generator and evaluator would be separate parties, so the full set of keys were not visible to the evaluator even though it remained available to the generator over the course of evaluation. In our setting, both generator and evaluator functions run on the same machine, so it is imperative that we make the full key set unavailable. We achieve this by instead supplying both the chosen subset of keys and the raw binary input of the client into the second phase of our modified *Battleship*. In this way, individual keys can be identified as either 0- or 1-keys without needing to examine the full key set.

D GC-based Case Study Setup

Client input. To arrive at a compact representation of Bob's input, we employ a simple bash shell script to:

- Remove unused chromosome and position fields.
- Remove comment lines at the start of file and line containing field headings.
- Remove the "rs" prefix from each SNP reference #.
- Remove all spaces between fields and line breaks between entries, making the entire input one line.
- Convert SNP reference numbers into hexadecimal format, and zero-pad the result to length 7 (hex format allows us to reduce 4 keys per entry for a more efficient representation).

- Merge Allele 1 and Allele 2 fields, and assign a 1-digit hex value to each possible allele pair.

The removal of all spaces and line breaks caters to the original *Battleship* design, which expects inputs to be read in as a single line. It is especially important that reference numbers be padded with zeroes (e.g., 0x3DE2 (15842), becomes 0x0003DE2), given that we merge all inputs into a single line, so entries can be parsed using fixed indices. 7 hex digits is sufficient to support all reference numbers, which have at most 8 decimal digits.

Vendor input. If a particular SNP has more than one ($allele\ pair$, $risk\ factor$) mapping, then each of these is treated as a separate input (with SNP reference number repeated). Although this leads to increased circuit size, specifying Alice's input in this manner is necessary in order to avoid subtle timing disparities which may leak information about the test being performed. The alternative is to make the if condition at line 10 of Algorithm 1 iterate over the mappings associated with each SNP. While it would result in less I/O time, fixing the loop bound according to the maximum number of mappings decreases performance if the majority of SNPs have few associated mappings. This would also complicate distinguishing between entries in our compact representation.

E Other Use Cases

Our OTP construction can also be adapted to other uses for one-time programs; we provide the intuition below. We must also consider the monetary costs associated with adapting programs into OTP boxes according to our design. If non-interactivity is not required, interactive garbled circuit protocols may suffice.

Additional genomic tests. Other tests are possible that operate on a sequenced genome. Further, Bob may have multiple inputs to evaluate on a single function. For instance, an individual may input two or more genomes for a paternity test or a disease predisposition test that may also involve other family members. This functionality can be easily added to the proposed scheme by treating multiple sets of test data as single input, although it does not provide privacy between family members (but provides privacy of the set from the vendor).

Temporary transfer of cryptographic ability. OTP lends itself naturally to the situation when one party must delegate to another the ability to encrypt/decrypt or sign/verify messages [17]. In this case, individual OTP boxes must be provisioned and given in advance to the designee, with each box only capable of performing a single crypto-operation. The cost could easily add up, but it might be acceptable for time-sensitive or infrequent messages of high importance (such as military communications). If messages are more frequent, then it may be worthwhile to consider a k-time extension ($k > 1$) to OTP. In either case, the designee is never given access to the raw private key. Care must be taken to restrict the usable time of each box, which can be realized by sealing an end date in addition to the one-timeness flag.

One-time proofs. As suggested by Goldwasser et al. [17], OTP allows witness-owners to go offline after supplying a proof token to the prover. This proof token can be presented to a verifier only once, after which it is invalidated. We can certainly realize this functionality using our OTP boxes, since proofs produced by our OTP are invalidated by nature of interactive proof systems and may not be reused. Depending on the usage environment, using our OTP box may or may not be cost-effective. While our implemented system may be too costly to serve as subway tickets, the cost may be justified if our box is used as an access-control mechanism to a restricted area.

Digital cash. As a one-time program, this was investigated by Kitamura et al. [34], which used Shamir's secret sharing in place of OTMs. We borrow their three-party scenario to reason about our own OTP system.

1. The bank supplies OTP boxes with set dollar values.
2. To make a payment, the user provides to the OTP box the shop's hash of a newly generated random number.
   - In TXT, the corresponding keys are selected.
   - After reboot, the selected keys are input into the garbled circuit program, which outputs a signature of the dollar-value concatenated with the shop's hash value.
3. The shop verifies the signature.
4. The shop requests cash from the bank using the signature.

Unlike Kitamura et al. [34], we have a proper OTM in the form of the TPM. A sealed flag value could enforce the one-timeness, preventing the user from giving valid signatures for more than one shop input. However, our scheme requires further modification to prevent double-spending, as it is possible for two independent shop hash-values to be the same, in which case the user can reuse the associated signature. Furthermore, OTP for digital cash would not be feasible if the held dollar value is less than the cost of the OTP box itself, unless the bank customer's goal is untraceability.

F Table of SNPs on BRCA1 and their risk factors

The genetic instructions that determine development, growth and certain functions are carried on Deoxyribonucleic acid (DNA) [57]. DNA is in the form of double helix, which means that DNA consists of two polymer chains that complement each other. These chains consist of four nucleotides: Adenine (A), Guanine (G), Cytosine (C), and Thymine (T). Genetic variations are the reason that approximately 0.5% of an individual's DNA is different from the reference genome.

SNP defines a position in the genome referring to a nucleotide that varies between individuals. Each person has approximately 4 million SNPs. Each SNP contains two alleles, which correspond to nucleotides

SNP Reference Number	Position	Alleles	Risk Factor
rs41293463	43051071	AT	6
		GG	6
		GT	6
rs28897696	43063903	AA	7
		AC	6
rs55770810	43063931	CT	5
		TT	5
rs1799966	43071077	GG	2
		AG	1.1
rs41293455	43082434	CG	5
		CT	5
		TT	2
rs1799950	43094464	GG	2
		AG	1.5
rs4986850	43093454	AA	2
rs2227945	43092113	GG	2
rs16942	43091983	AG	2
		GG	2
rs1800709	43093010	TT	2
rs4986852	43092412	AA	2
rs28897672	43106487	GG	4
		GT	4

Table 7. SNPs on BRCA1 and their corresponding risk factors for breast cancer.

Table 7 lists the SNPs related to BRCA1 and the corresponding risk factors. The magnitude of risk factors ranges from 0 to 10 [54]. A risk factor greater than 3 indicates a significant contribution of that particular allele combination to the overall risk of contracting breast cancer.

G Genomic Algorithm

The genomic algorithm is shown in Algorithm 1. Here, RF corresponds to the total risk factor for developing breast cancer. The "risk factors" file contains the associations between the observed SNP and the risk factor, while the "patient SNPs" file contains a patient's extracted SNPs.

H Porting effort

To get started with an application based on our OTP (either variant), the very first step is always creating the payload program, or if existent, porting it to a designated programming language. In the case of the GC-based, rewriting in the Frigate-specific limited-C language is necessary. For the TXT-only, the few technical tweaks such as PATA I/O with our added DMA support are seamlessly transparent to the application developer, since they are only exposed like POSIX-like file operations, similar to *fopen*, *fread* and *fwrite*. We argue that regarding

Algorithm 1 Genomic Algorithm

```
Input: $RiskFactors, Patient\ SNPs$
Output: $RF$
 1: procedure GENOMIC ALGORITHM($RiskFactors, Patient\ SNPs$)
 2:     $RF = 0$
 3:     for line in Patient SNPs do
 4:         SNP_ID = SNP ID in line
 5:         ALLELES = two alleles in line
 6:         for line_rf in Risk Factors do
 7:             SNP_ID_rf = SNP ID in line_rf
 8:             ALLELES_rf = two alleles in line_rf
 9:             if SNP_ID = SNP_ID_rf then
10:                 if ALLELES = ALLELES_rf then
11:                     RF = RF + risk factor in line_rf
12:                 else
13:                     RF = RF + 0
14:                 end if
15:             else
16:                 RF = RF + 0
17:             end if
18:         end for
19:     end for
20:     return $RF$
21: end procedure
```

the status quo of most existing OTP solutions, this process has to be (quasi-)manual in terms of programming language. Therefore, we hope, as future work, to either have an automated framework for OTP-specific conversions or (in the case of TXT-only) include a lightweight language environment.

I SMM attacks and TPM relay attack

SMM attacks. The System Management Mode (SMM) is a special execution mode in modern x86 CPUs and considered having (informally) the Ring minus 2 privilege, preempting virtually any other modes. Therefore, although not recently, it was exploited [68] to interfere with TXT execution. This attack assumes the compromise of the SMI (SMM Interrupt) handler (which is difficult, but feasible in an ad-hoc manner), and during TXT execution an SMI is triggered and the compromised handler comes in to manipulate anything of the adversary's choice. However, in the case of our OTP, we do not load any standard code that needs SMI and has it enabled (like an OS or hypervisor); instead, our custom program for key selection or OTP execution has SMI disabled from the first line of code (not to mention containing any SMI trigger, e.g., writing to port 0xb2), and thus is not affected by such attacks. Since TXT is exclusive, no other code can run in parallel.

Note that Alice no longer benefits from any attacks (e.g., stealing Bob's input) due to loss of physical possession and network connectivity. We exclude, for now, any potential threats from Intel ME (Management Engine) which is referred to as Ring minus 3 and has a dedicated processor, in the consideration that all rely on ad-hoc vulnerabilities and this topic is still under open discussion [13].

TPM relay attack [14]. A man-in-the-middle (MitM) attack specifically targeting TPM-like devices impersonates and forwards requests to a (remote) legitimate device, pretending its proximity or co-location on the same machine, to either learn the secrets or forge authentication/attestation results. In the case of our OTP, only Bob has physical possession and is motivated for such attacks. However, since he cannot clone the TPM chip, whatever real traffic directed to the legitimate one will cause irreversible effect (e.g., flipping the flag) Note that his intension is not merely mimicking, which does not help. Also, we do not send TPM commands in plaintext, except for ordinals and certain metadata. Our ultimate argument is that, regardless of the lab effort we already exclude in Section 2.2, the integration of TPM in other microchips (e.g., SuperIO) or an equivalent method will avoid exposing TPM pins for potential probing.