Mitigating Password Database Breaches with Intel SGX

Helena Brekalo
iMinds - DistriNet, KU Leuven
3000 Leuven
Belgium
helena.brekalo@
student.kuleuven.be

Raoul Strackx
iMinds - DistriNet, KU Leuven
3000 Leuven
Belgium
raoul.strackx@
cs.kuleuven.be

Frank Piessens
iMinds - DistriNet, KU Leuven
3000 Leuven
Belgium
frank.piessens@
cs.kuleuven.be

ABSTRACT

In order to prevent rainbow attacks against a stolen password database, most passwords are appended with a unique salt before hashing them as to make the password random and more secure. However, the decreasing cost of hardware has made it feasible to perform brute force attacks by guessing the passwords (even when extended with their salt).

Recently Intel has made processors with Intel SGX commercially available. This security technology enables developers to (1) completely isolate code and data running in an SGX enclave from untrusted code running at any privilege layer and (2) prevent data sealed to an enclave from being accessed on any other machine.

We propose to add a key to the password (and salt) before they are hashed. By calculating the hash within an enclave, the key never leaves the enclave. This provides much stronger protection; offline attacks are infeasible without knowledge of the key. Online attacks on the other hand are much easier to defend against.

CCS Concepts

•**Security and privacy** → **Authentication; Software security engineering;** *Hardware-based security protocols;*

Keywords

Intel SGX, Password Security

1. INTRODUCTION

Just a week after Dropbox required its users to reset their passwords after a security breach in 2012, Spotify is now asking the same of theirs. Anybody who regularly checks technology websites is familiar with such news stories. Specialized websites such as Have I Been Pwned[1] show that passwords are often compromised, among other personal information. This is worrisome as users often reuse the same password for multiple services.

As a defense-in-depth approach against such breaches, passwords should never be stored in clear text. Unfortunately, storing only hashed versions of users' passwords does not stop persistent attackers. The very nature of passwords and the breached data means that strong security properties can no longer be guaranteed. Attackers are left with various options. They could, for example, exploit the knowledge that users often use easy to guess passwords and a large amount of passwords can be tried against the captured password database. To make matters worse, the cost to calculate a hash function has steadily dropped due to Moore's law, and development of specialized hardware (e.g., FPGAs) and software (e.g., to optimize Bitcoin mining software). Adding a salt (i.e., a randomly generated nonce) before hashing does not prevent such attacks; with the breached password database, the salts are likely to be compromised as well.

Various alternatives have been proposed that require an attacker to increase their computational power [13] or memory storage [10] capabilities. However, such solutions have limited impact against sophisticated attackers and increase the burden on defenders as well; service providers often need to be able to handle a large amount of concurrent access requests. The performance or memory overhead that can be accepted to provide stronger protection of a breached password database, is thus limited.

The only strong security solution is to keep a part of the hash composition secret, even when the machine itself is compromised. Such a requirement could be met with dedicated hardware security modules (HSMs). Unfortunately such devices can be costly, especially when they need to be able to support a large volume of login requests per second. Moreover, industry is migrating quickly to the cloud. Such an environment provides additional challenges. Not only do the same security problems exist in the cloud, virtualized machines must also be migratable to different physical machines. The recent Snowden revelations also sparked worries about physical machines located in foreign countries and potential government subpoenas. This places additional challenges on hardware security modules.

Intel Software Guard Extensions (Intel SGX), new technology added to Intel processors, presents an interesting new direction. Instead of relying on a physical hardware security module, Intel SGX could be used to implement an HSM in software with similar security guarantees [17]. This would effectively reduce the capabilities of an attacker who breached the (hashed) password database from an offline attacker to that of an online attacker; attacks against user credentials remain possible but attackers need to be able to directly ac-

[1] haveibeenpwned.com

SysTEX '16, December 12-16 2016, Trento, Italy

ISBN 978-1-4503-4670-2/16/12...$15.00

DOI: http://dx.doi.org/10.1145/3007788.3007789

cess the security module. It is much easier to defend against such online attackers.

We make the following contributions:

- We propose a novel design to protect stored user credentials against offline attacks.
- We discuss how enclaves can be migrated with their virtual machines between different physical servers. We present an extension of Intel SGX's attestation service to ensure that enclave instances will only be moved to physical machines of the cloud provider.
- We implemented and evaluated our design. Microbenchmarks show that performance overhead is limited and likely negligible when applied in the wild.
- We evaluate our approach against known and predicted attacks against SGX.

The remainder of this paper is structured as follows: In Section 2 we discuss in detail the attack model we assume and the security properties we wish to guarantee. After giving a brief introduction to Intel SGX in Section 3, we present our solution in Section 4. Its implementation is discussed in Section 5 and evaluated in Section 6. We finish with an overview of related work and a conclusion.

2. PROBLEM STATEMENT

We assume the following setting: An enterprise provides a service to a large number of clients and requires them to authenticate before they are allowed to access its service. The enterprise outsourced its infrastructure to a cloud provider; its services are virtualized and the cloud provider may decide to move them from one physical service to another at any moment in time. The cloud provider's physical servers are equipped with SGX-enabled hardware.

We assume that an attacker (or malicious cloud provider) may be able to launch sophisticated attacks against the enterprise's servers. Such attacks may succeed to (1) gain in-kernel access and (2) exfiltrate the password database.

Under such conditions, we wish to guarantee that an attacker is computationally only as powerful as an online attacker. Bruteforce attacks against the leaked password database may still be launched, but their chances of success are equivalent to inverting the used cryptographic hash function.

Online attacks are much easier to defend against than offline attacks. This is a well-researched topic, and includes mitigations such as forcing users to choose passwords with enough entropy, automatically lock out users indefinitely [14, 19, 21] or for a period of time to defend against bruteforce attacks, etc. Such mitigations are important, but orthogonal to the problem we wish to solve.

For cryptographic primitives, we assume the Dolev-Yao model. Cryptographic messages may be replayed, but it is infeasible to invert hash functions or find a collision.

3. BACKGROUND: INTEL SGX

Providing strong security guarantees on commodity hypervisors and kernels is challenging. With a trusted computing base of multiple millions lines of code, the presence of previously undetected vulnerabilities is almost a mathematical certainty. Additionally, the construction of this software poses additional risks. Their single address space implies that once an attacker has managed to exploit a single vulnerability, the entire hypervisor or kernel is compromised. As they assume that lower-level layers are implemented perfectly, successful exploitation of a vulnerability in such a lower-level layer enables an attacker to attack any software component running on top.

In 2008 McCune et al. [11] proposed a radically different, non-hierarchical design. Instead of relying on a layered approach, security sensitive pieces of applications should be completely isolated from the rest of the system. This idea was coined a protected-module architecture (PMA). Since Intel implemented such a PMA in their Skylake processors, research interest has sparked.

Protected-module architectures rely on two basic properties. First, security sensitive code and data can be completely *isolated* from any other part of the system. Access to such "modules" (or "enclaves" in SGX-terminology) is protected with a program-counter-based access control mechanism [22]: When code is running outside of the module, access is heavily restricted. Only when the module is entered through specially created entry points does the program counter point within the module, and can its stored code and data be accessed.

Related work [1, 2, 15, 16] showed that this access control mechanism enables protected modules to provide strong security guarantees when (1) this isolation mechanism is implemented perfectly, (2) sufficient security checks are taken when a module is called and (3) the module itself is free from vulnerabilities. Unfortunately, recent results show that Intel SGX (and probably most/all other protected-module architectures [6, 23]) currently do not provide perfect isolation. Design choices (e.g., a malicious or badly written enclave should never cause the system to stop responding) lead to side channels that allow attackers to largely reconstruct the control flow graph of the enclave. To prevent such attacks, enclaves should be written carefully; developers should avoid that the code branches on secret information.

Second, protected-module architectures rely heavily on *key derivation*. By deriving a cryptographic key based on the initial state of the module (i.e., its contents and size) and a unique platform secret, each module can easily access its own, unique cryptographic key. Access to this key is subject to the same program-counter-based access control. Only the currently executing protected module may directly access its own key.

Key derivation is an interesting primitive as it enables protected modules to use their own key to confidentiality, integrity and version protect sensitive data for the next invocation. Similarly, the same construct can be used for attestation purposes. In many settings a user should be able to cryptographically prove to a remote party that it executed the correct protected module, with the presented input leading to the returned result.

Intel SGX also provides protection against sophisticated hardware attacks. It does so by guaranteeing that enclave code and data only reside in plaintext within the CPU package (e.g., the processor's cache). Whenever contents need to be sent to main memory, it is confidentiality, integrity and version protected. Thus an attacker executing a cold boot attack [9], or who snoops the system bus, cannot extract sensitive data stored within the enclave's boundaries.

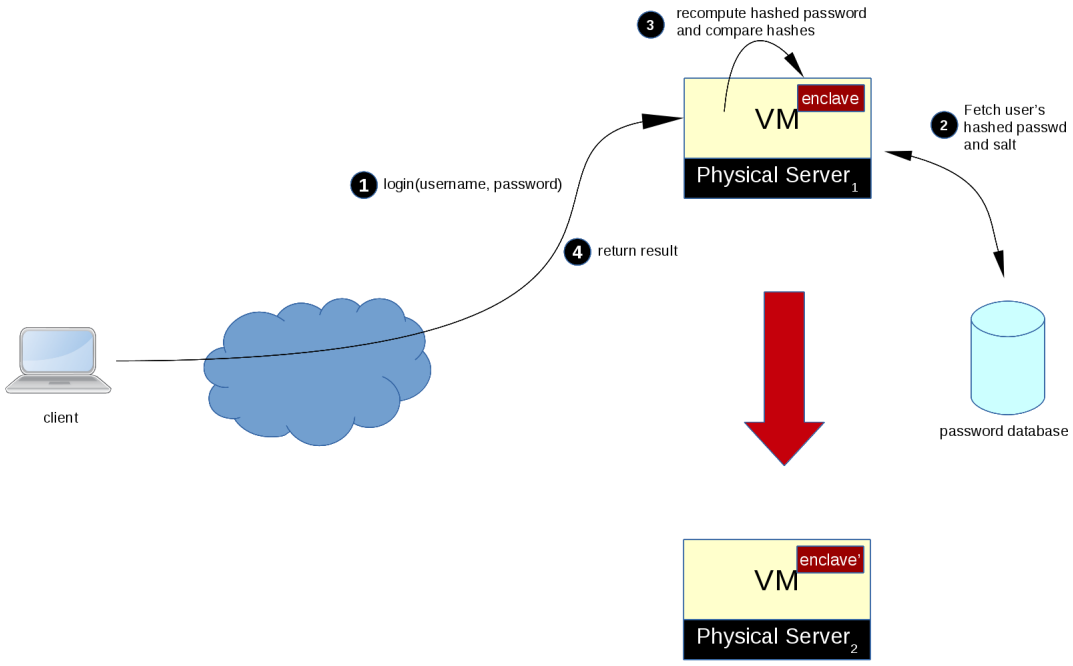

Figure 1: Graphical overview of how a user's password is checked. As in a cloud environment virtual machines may be moved between physical machines, SGX enclaves need to be migratable as well.

4. PROTECTING PASSWORDS AGAINST OFFLINE ATTACKS

An enterprise that wishes to take advantage of her cloud provider's SGX-enabled hardware needs to tackle two problems: Making optimal use of SGX's isolation mechanism to protect users' passwords and dealing with the cloud provider's abilities to move virtual machines. We tackle both problems sequentially.

Checking a client's password.

In order to provide stronger protection against an attacker who launches (offline) bruteforce attacks against a breached password database, we propose to store the passwords as the result of an HMAC function with a cryptographic key k:

$$\text{password}_{\text{stored}} = \text{HMAC}(k, \text{password}||\text{salt})$$

where "||" represents string concatenation.

In order to keep the cryptographic key k secure, we place it in an enclave. Only two entry points to the enclave are provided: (1) given a user's password and salt, the HMAC is calculated and returned and (2) another to facilitate resuming the enclaves' execution on a different physical machine.

Figure 1 shows the operation graphically. First, a user connects to the enterprise's server and provides its username and password. Next, the server looks up the user's stored password and salt in the password database (potentially located on a different machine). The salt is provided to the password enclave together with the user-provided password. There the resulting HMAC is calculated. The result is compared to the reference password from the password database. When both match, the user is logged in. Otherwise an error is returned.

Dealing with VM migration.

Building the HMAC-calculating enclave is simple but dealing with VM migration is more challenging: SGX prevents enclaves from being migrated directly with their VM image. Enclaves need to be recreated at their destination. As all user passwords are protected with the same cryptographic key k, this key needs to be transferred to the new physical machine in a secure way.

There are various options to achieve this. First, we could set up a secure communication channel between the previous physical machine and the new one. Intel SGX's attestation features guarantee that an attacker cannot not intercept k in any way. We presented this approach in previous work [20]. Unfortunately, this approach is suboptimal in this case. In order to set up a communication channel, the virtual machine images must be live at both communication end points. Afterwards the cloud provider should be signalled that the old one can be destructed. This may require changes to the provider's management software.

Second, a dedicated server could be created to distribute k to a newly created or migrated virtual machine. As in the previous option, SGX's attestation services can provide a secure channel. As with any central server, availability may be of concern. Alternatively, virtual machines could also establish a peer-to-peer network of virtual machines, where each one can choose to distribute k to newcomers.

In each case a secure channel is created between two physical machines. Intel SGX's attestation features guarantees that both platforms are genuine Intel platforms. However, additional checks should be added to ensure that enclaves are only migrated to physical machines of the cloud provider. An attacker who manages to move an enclave to her own machine has indirect access to k. Adding security measures within the enclave would only partially solve the problem.

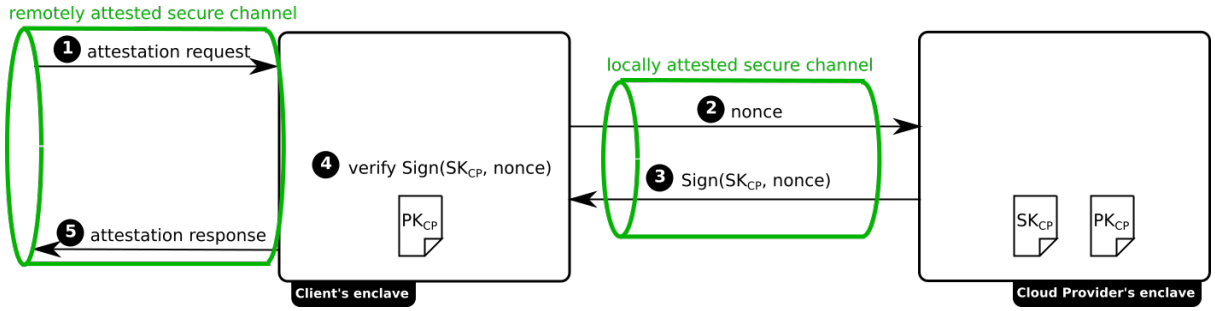

Figure 2: In order to prevent the password enclave from being migrated to the attacker's device, we need to guarantee that enclaves are only migrated to physical machines belonging to the cloud provider. A simple, non-migratable cloud provider's enclave can provide the required guarantees during remote attestation.

Consider as an example an enclave that rate limits the number of guesses an attacker may execute. While this may be implemented effectively for each enclave, an attacker would still have the ability to deploy an almost unlimited number of identical, concurrently executing enclaves. Enforcing guess rate limits at the overall enterprise level would likely cause management problems and deteriorate performance significantly.

We believe that such considerations are an important, general limitation of using Intel SGX in a cloud environment. We propose a solution where the cloud provider runs a (non-migratable) "cloud provider enclave" on all of its physical servers. During attestation of clients' enclaves, they should verify that they execute on the same physical platform as the cloud provider's cloud enclave.

To achieve this, we assume that each instance of the cloud provider's enclave on each physical machine has access to the cloud provider's public-private (PK_{CP} & SK_{CP}) key pair; both parts are sealed to the enclave's identity and stored locally by the hypervisor. Enclaves that wish to verify that they execute on one of the cloud provider's physical machines are linked with its public key PK_{CP}.

Figure 2 displays our approach graphically. In the first step, a client enclave is deployed on the platform (e.g., the password enclave) and receives an attestation request.

In step two, the client establishes a secure, local communication channel with the cloud provider's enclave [3] and a nonce is exchanged. SGX's attestation features already differentiate local from remote attestation; only when both enclaves run on the same physical machine will a secure communication channel be established.

In the third step, the cloud provider's enclave signs the received nonce with its private key SK_{CP} and returns it to the client's enclave.

Next, the client enclave verifies in step four that the nonce is signed with the provider's private key. As it is already verified in step 2 that it runs on the same physical machine as the provider's enclave, it must now execute on genuine physical hardware of the cloud provider. Now it only needs to send a positive reply to the initial (remote) attestation request from step 1.

5. IMPLEMENTATION

We implemented a proof-of-concept of our approach. The enclave supports password verification, but migration to different physical machines is left for future work.

Our implementation uses an HMAC function based on the new SHA-3 standard, downloaded from the Keccak page [8]. The used key k and salts are generated using the hardware `rdrand` instruction.

Table 1 shows the number of lines of code executing inside the enclave. The total enclave is implemented in 2,012 LoC. The SHA-3 implementation consumes the largest part with 1,805 LoC. Our own HMAC implementation – excluding the SHA-3 implementation – takes 34 LoC and is part of the enclave, so it is not counted in the total lines of code. These measurements were performed using David A. Wheeler's 'SLOCCount'.

	Lines of Code
Enclave	173
HMAC	34
SHA-3	1,805
Total lines of code	2,012

Table 1: Number of lines of code for different parts of the enclave.

6. EVALUATION

Performance evaluation.

We microbenchmarked our solution by computing 10,000 passwords. The tests were performed on a laptop with an Intel Core i7-6500U CPU running at 2.50 GHz with 8 GB of RAM and making use of version 0.10 of the SGX driver[2] and SDK version 1.5[3]. The median of the results of these tests can be found in the Table 2. When the passwords were HMAC'ed within the enclave, it took 0.046 ms per password. We compared this to only computing a SHA-3 hash of an equally long salted password in unprotected memory. With only 0.007 ms/password this approach was considerably faster, but passwords would be relatively easy to bruteforce in practice.

We attribute the performance difference to the fact that crossing the enclave's borders is a relatively costly operation compared to the cheap SHA-3 operation. Also note that the HMAC internally uses two SHA-3 calls.

[2]From https://github.com/01org/linux-sgx-driver
[3]From https://github.com/01org/linux-sgx

With an overhead of 0.039ms/password, we believe that these microbenchmarks show that our approach is practically feasible. In future work we plan to implement a complete setup where a user interacts with an Apache server and a MySQL database to log in to a webpage. Based on these microbenchmarks, we suspect that the incurred practical overhead is well below alternative approaches applied in practice.

	SGX	No SGX
Time (ms)	0.046023	0.006788

Table 2: Performance measures of the creation of passwords with and without the use of SGX.

Security evaluation.

With the use of SGX, different attack scenarios for our implementation are impossible.

An attacker may attempt to derive the secret key from a stored password. Such attacks are impossible against our implementation as it requires inverting the HMAC algorithm. This means that even if an attacker creates a user account for himself and knows the resulting hash, he will not be able to derive the secret key that was used.

This is why the use of a salt is still needed: if an attacker were to know the resulting hashes for certain passwords and no salt would be added, it would be possible to derive passwords that result in the same hash within the system. This would allow her to breach user accounts, even without knowing the secret key. If a random salt is appended to each user's password, the password becomes truly unique and knowing a password that results in the same hash will not suffice to break into a user's account (since the salt of the two accounts will differ).

In case an attacker has access to the virtual machine that manages the passwords, he can use brute force to guess user passwords, but he will need continuous access to the machine and could be detected by various detection mechanisms such as an IDS. Such online attacks can also be mitigated by forcing users to make use of sufficiently strong passwords or by maintaining a rate limit where the number of guesses for a user password is limited to a pre-defined number of guesses.

Our solution does not provide protection against replay attacks of the hash: if an attacker manages to get inside the database and get hold of previous secrets of a user, these can be reinstantiated, granting the attacker access to this user's account. Mitigating such rollback attacks is feasible [21], but left for future work.

Attackers may also launch side-channel attacks against the password-calculating enclave. In an execution-time side-channel attack, an attacker tries to determine how long the enclave takes to execute a calculation by providing different input. By analyzing timing differences for different inputs, an attacker can gain insight into the workings of the system and extract sensitive data. This attack is not possible because our solution does not branch on the secret key nor the provided password.

Xu et al. [23] recently showed that when an attacker pages out enclave pages, sensitive data may be leaked. Similarly, as we don't branch on a secret, such attacks are not possible against our implementation.

A cold-boot attack is another side-channel attack scenario. In such an attack, the user's drive is removed from the system while it was up and running and inserted into an attacker-controlled system [9]. With memory protection no longer being enforced by the processor, sensitive data is readily accessible to the attacker. Because SGX encrypts data travelling between the system cache and memory, this attack is not possible. This defense mechanism also prevents bus snooping attacks.

7. RELATED WORK

With the arrival of time-sharing machines in the 1960s, a need arose to authenticate users. Passwords are an obvious solution and techniques to store them have been researched ever since. We categorize them according to the mechanism they rely on to secure passwords.

In very early systems such as the CTSS time sharing system, passwords were stored in plaintext. Bugs in such a design may inadvertently leak passwords to benign users, and they did [5]. To avoid such failures, passwords were later hashed.

When the cost of storage went down, large databases of hashed passwords could be pre-generated and used to lookup matching passwords for a given hash. To make such attacks harder, a salt was added at the time the passwords were hashed.

Salts only prevent rainbow table attacks against password databases. Unfortunately, with the performance overhead of a single hash function dropping steadily over time – especially when specialized hardware is used such as a GPU or FPGAs – bruteforce attacks have become feasible again.

To prevent such attacks, new algorithms have been developed to artificially increase the time required to compute the resulting hash. PBKDF2 [13] for example can be configured to hash a password not once, but an arbitrary number of times. The disadvantage of such mechanisms is that the defender is slowed down as well.

Scrypt [10] takes a different approach. Instead of artificially decreasing the performance of a hash calculation, the algorithm attempts to increase the amount of memory that is required during its execution. Unfortunately, this too decreases the number of authentication request per second that a defender can support.

In addition to the use of stronger cryptographic primitives, password handling could also be done isolated [7] from the generic web server. Big companies such as Facebook [12] take this approach and employ a completely physically isolated machine. As these servers still rely on a commodity operating system, their TCB is considerable. By relying on Intel SGX, we avoid a huge TCB.

More recently, special hardware security modules (HSMs) are being considered to store passwords and/or cryptographic keys. The Yubikey for example can be used by users to generate one-time-passwords. For websites that do not support OTPs, the Yubikey can store static passwords.

On the server-side, hardware security modules can be used to store user credentials as well. Unfortunately, they are often expensive. In 2012 Graham Steel proposed an alternative [18]; keep a local parameter secret and stored within a low-cost peripheral. While the peripherals can be audited – in contrast to many high-end HSMs – the peripherals cannot defend against a hardware-based attacker.

Himanshu et al. [17] showed that such HSMs could also be

implemented in software. Their industry-application implements a software-based TPM chip running on top of ARM TrustZone. They also discussed options to port their fTPM to Intel SGX, but face additional architectural challenges.

Finally, Joseph Birr-Pixton presented a proof-of-concept to check passwords in a blog post [4]. They use a different approach: Passwords are passed to the PBKDF2 function and the result is encrypted with a region key. This results in a performance overhead of more than 3 orders of magnitude larger than our HMAC-based approach. In addition, we propose a solution to (automatically) migrate the password enclave, making it feasible in a cloud setting. We also provide a detailed security analysis.

8. CONCLUSION

Powerful hardware and custom designed hardware (e.g., GPUs and FPGAs) have become affordable, making it possible for attackers to perform bruteforce attacks against breached password databases. Adding a salt to a password is no longer enough to enforce its security. In this paper we propose the use of SGX to isolate a secret key and to include it in the password hash. Because part of the hashed message is secret, it becomes computationally infeasible to bruteforce the passwords. Our solution is intended for use in a cloud environment and makes use of the HMAC hashing algorithm and SHA-3. Despite a limited performance overhead, we believe this work could be the basis for future work to enhance password security.

Acknowledgments

Raoul Strackx holds a Postdoctoral mandate from the Research Foundation Flanders (FWO). This research is partially funded by the Research Fund KU Leuven, and by the Research Fund - Flanders (FWO).

9. REFERENCES

[1] P. Agten, B. Jacobs, and F. Piessens. Sound modular verification of C code executing in an unverified context. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'15)*, Jan. 2015.

[2] P. Agten, R. Strackx, B. Jacobs, and F. Piessens. Secure compilation to modern processors. In *Proceedings of the 25th Computer Security Foundations Symposium*, CSF'12, 2012.

[3] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP'13. ACM, 2013.

[4] J. Birr-Pixton. Using SGX to harden password hashing, January 2016.

[5] F. Corbato. On building systems that will fail, 1990.

[6] V. Costan and S. Devadas. Intel SGX explained. Cryptology ePrint Archive, Report 2016/086, 2016.

[7] A. Everspaugh, R. Chaterjee, S. Scott, A. Juels, and T. Ristenpart. The pythia PRF service. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Aug. 2015.

[8] M. P. Guido Bertoni, Joan Daemen and G. V. Assche. The keccak sponge function family, 2016.

[9] J. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Appelbaum, and E. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*. ACM, 2008.

[10] Internet Engineering Task Force (IETF). The scrypt password-based key derivation function, Aug. 2016.

[11] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for TCB minimization. In *Proceedings of the ACM European Conference in Computer Systems*, EuroSys'08. ACM, Apr. 2008.

[12] A. Muffett and A. Bajenov. Facebook: Password hashing & authentication, December 2014.

[13] Network Working Group. PKCS #5: Password-based cryptography specification (RFC 2898), Sept. 2000.

[14] B. Parno, J. R. Lorch, J. R. Douceur, J. Mickens, and J. M. McCune. Memoir: Practical state continuity for protected modules. In *Proceedings of the IEEE Symposium on Security and Privacy*, S&P'11, pages 379–394, May 2011.

[15] M. Patrignani, P. Agten, R. Strackx, B. Jacobs, D. Clarke, and F. Piessens. Secure compilation to protected module architectures. In *Transactions on Programming Languages and Systems (TOPLAS)*, volume 37, pages 6:1–6:50. ACM, Apr. 2015.

[16] M. Patrignani, D. Clarke, and F. Piessens. Secure Compilation of Object-Oriented Components to Protected Module Architectures. In *Proceedings of the 11th Asian Symposium on Programming Languages and Systems (APLAS'13)*, volume 8301, 2013.

[17] H. Raj, S. Saroiu, A. Wolman, R. Aigner, J. Cox, P. England, C. Fenner, K. Kinshumann, J. Loeser, D. Mattoon, M. Nystrom, D. Robinson, R. Spiger, S. Thom, and D. Wooten. fTPM: A software-only implementation of a TPM chip. In *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, Aug. 2016. USENIX Association.

[18] G. Steel. Mac in the box, 2012.

[19] R. Strackx, B. Jacobs, and F. Piessens. ICE: A passive, high-speed, state-continuity scheme. In *Annual Computer Security Applications Conference*, ACSAC'14, 2014.

[20] R. Strackx and N. Lambrigts. Idea: State-Continuous Transfer of State in Protected-Module Architectures. In F. Piessens, J. Caballero, and N. Bielova, editors, *Engineering Secure Software and Systems*, volume 8978 of *Lecture Notes in Computer Science*, pages 43–50. Springer International Publishing, Mar. 2015.

[21] R. Strackx and F. Piessens. Ariadne: A minimal approach to state continuity. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 875–892, Austin, TX, Aug. 2016. USENIX Association.

[22] R. Strackx, F. Piessens, and B. Preneel. Efficient Isolation of Trusted Subsystems in Embedded Systems. In *Security and Privacy in Communication Networks (SecureComm'10)*, volume 50, 2010.

[23] Y. Xu, W. Cui, and M. Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *36th IEEE Symposium on Security and Privacy*. IEEE, May 2015.