Security challenges and opportunities in adaptive and reconfigurable hardware

Security Challenges and Opportunities in Adaptive and Reconfigurable Hardware

Victor Costan and Srinivas Devadas
Computer Science and Artificial Intelligence Laboratory (CSAIL)
Massachusetts Institute of Technology, Cambridge, MA
victor@costan.us, devadas@csail.mit.edu

***Abstract*—We present a novel approach to building hardware support for providing strong security guarantees for computations running in the cloud (shared hardware in massive data centers), while maintaining the high performance and low cost that make cloud computing attractive in the first place. We propose augmenting regular cloud servers with a Trusted Computation Base (TCB) that can securely perform high-performance computations. Our TCB achieves cost savings by spreading functionality across two paired chips. We show that making a Field-Programmable Gate Array (FPGA) a part of the TCB benefits security and performance, and we explore a new method for defending the computation inside the TCB against side-channel attacks.**

I. Introduction

Cloud computing can bring great cost reductions by leveraging economies of scale inherent in data center setup. However, the security implications of executing sensitive computations on shared hardware make cloud computing infeasible for applications such as medical and financial data processing.

Current "cloud" infrastructures are not secure because:

- The cloud provider has access to all the computation and data on a cloud server.
- A successful hypervisor attack can expose all computation and data to other parties sharing the hardware.
- Information leaked via side channels is available to the cloud provider, as well as to any party sharing the hardware. For example, [1] outlines how shared caches can be exploited to extract RSA and AES keys in Amazon's cloud.

We propose solving the challenges above with specially designed hardware, packaged as an add-on for cloud servers (figure 1), like nVidia's Tesla GPU cards. Cloud applications would separate the computation logic, targeting the secure hardware, from the code responsible for I/O and coordination, which runs on untrusted server hardware, such as x86 processors. This seems like an extra burden for developers, but we note that logical separation of I/O from computation is a prerequisite for their overlapped execution, which is in turn necessary for achieving high-performance in regular applications [2]. Section II provides an example of logic separation for the case of secure cloud storage.

This paper introduces three major insights for implementing the secure add-on. Section III shows how to reduce the TCB's cost by splitting its functionality into a $\mathcal{P}$ (processing) chip with high processing power and volatile memory, and an $\mathcal{S}$ (state) chip with secure NVRAM. Section IV argues that FPGAs are excellent candidates for the $\mathcal{P}$ chip, and discusses using their reconfiguration ability for secure high-performance computation. Section V explores a method of protecting against timing and power side-channel attacks that is applicable to many classes of sensitve computation.

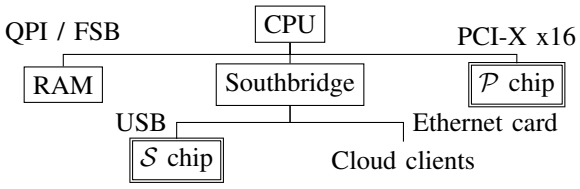

Fig. 1. Cloud server augmented with a TCB. TCB components (the $\mathcal{S}$ and $\mathcal{P}$ chips) have double borders. Everything else is untrusted.

A. Chain of Trust

We provide an overview of the chain of trust used to assert the security of computation peformed on a remote server. We closely follow the Trusted Platform Model (TPM) [3].

At manufacturing, the $\mathcal{S}$ chip generates an endorsement key pair $(\mathsf{PubEK}, \mathsf{PrivEK})$. PubEK is signed by the manufacturer, producing the chip's endorsement certificate (ECert). The manufacturer acts as a CA, and its certificate is a promise that the associated PrivEK will not be revealed outside the $\mathcal{S} - \mathcal{P}$ chip pair. (The pairing of these chips is a subject of Section III.)

When the cloud server connects to a client, it presents its ECert. The client software checks the CA key against its trusted list, and uses PubEK to encrypt a session key, which becomes the shared secret between the client and the trusted hardware on the server.

II. Example: Secure Cloud Storage

This section provides an example of breaking up an application into a component that runs on trusted hardware, and a component that runs on untrusted hardware. We focus on implementing a single application, secure storage, but the approach presented here generalizes to other high-performance applications.

For brevity, we assume a single-server system that exposes a single large virtual disk to all its clients via a block-oriented API. Our simplifications are acceptable because all well-known file systems are layered on top of block devices.

We discuss the implications of providing integrity guarantees, focusing on freshness (reads reflect the most recent write), which is impossible to do with untrusted software alone [4].

The storage application runs controller code responsible for network communication, disk I/O, and scheduling on an untrusted platform, and relies on the TCB to guarantee integrity and freshness.

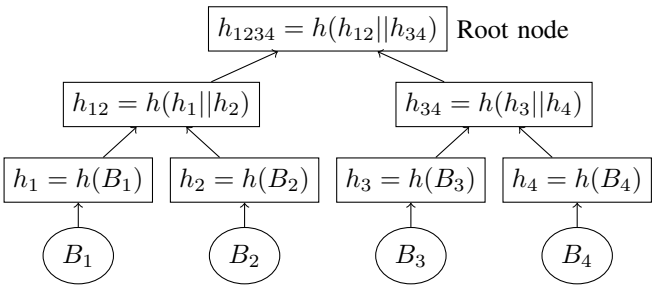

Fig. 2. Merkle Tree for a disk with 4 blocks. Tree leaves contain the cryptographic hashes of corresponding blocks. Inner nodes hash the contents of their children. The root node's hash can be used to recursively verify the entire tree, and its content is stored in the $\mathcal{P}$ and $\mathcal{S}$ chips.

The system guarantees integrity and freshness by maintaining a Merkle tree [5] (figure 2) over the data blocks. The tree content updates are computed by the $\mathcal{P}$ chip. The tree's root hash is stored in the $\mathcal{P}$ chip's SRAM during a power cycle, and in the $\mathcal{S}$ chip across power cycles. For performance, the $\mathcal{P}$ chip also caches tree nodes. The caching strategy is implemented in untrusted software running on the server OS.

The $\mathcal{P}$ chip is trusted to verify the clients' read operations using the Merkle tree, and to update the tree to reflect write operations. The $\mathcal{P}$ chip caches tree nodes in its SRAM. The cache policy is implemented in the controller, and can be changed quickly to reflect the client applications' access patterns. This section describes the cache management protocol and argues that a malicious controller cannot impact the freshness and integrity guarantees.

Entry	Node	Hash	Verified	Left	Right
0	1	cdaf...	Y	Y	Y
1	5	a1b2...	Y	N	N
2	2	e935...	Y	Y	Y
3	4	b54c...	Y	N	N
4	7	9348...	N	N	N
5	3	71f3...	Y	N	N
6	6	bb72...	N	N	N

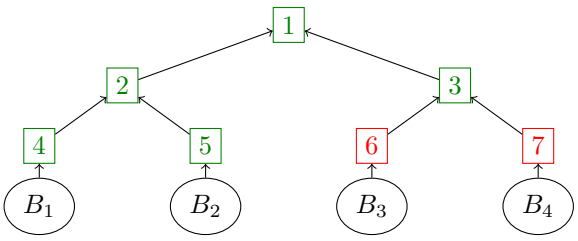

Fig. 3. $\mathcal{P}$'s cached view of the Merkle tree. Nodes 6 and 7 are loaded in the cache, but not verified, so the $\mathcal{P}$ chip cannot be used to sign reads for B_3 and B_4. It can sign reads for B_1 and B_2.

The SRAM cache is made up of entries mapping to Merkle tree nodes (figure 3). An entry contains the node number[1], the hash stored in the node, and the following status bits:

- V is set if the node's hash has been verified to be correct.
- L and R are set if the node's left, respectively, right child is loaded in the cache and its V line is correct.

To guarantee freshness, the $\mathcal{P}$ chip ensures that a node is stored in the cache at most once. To achieve this without a full search on each insertion, we don't allow evicting nodes whose children are cached. L and R are used to enforce this restriction.

Therefore, the controller **loads** an entry by providing the new node number and hash, as well as the entry holding the parent of the node being evicted. The $\mathcal{P}$ chip uses this to clear the parent's L or R flag, if the evicted node was verified. After a load, the controller **verifies** an entry by providing numbers for the entries holding the node's parent and sibling. The $\mathcal{P}$ chip verifies that the parent's V flag is already set, and that L and R flags match the children's V flags, to avoid duplicate entries for a node. If verification succeeds, the V flag is set on the children entries, and L and R are set on the parent entry.

Clients are assured of the correctness of their operations by HMACs produced by the $\mathcal{P}$ chip. For a **read**, the controller supplies the $\mathcal{P}$ chip with the entry holding the block's Merkle leaf. The $\mathcal{P}$ chip checks that the entry has V set, and produces an HMAC over the block number, block hash, and a client-generated nonce. When **writing** a block, the controller provides the entry numbers for all the nodes on the path from the block's Merkle leaf to the root, as well as the entries for the nodes' siblings. The $\mathcal{P}$ chip checks that all the entries are verified, and verifies that the nodes make a path from the leaf to the root. If the checks (which are critical to the freshness guarantee) succeed, the $\mathcal{P}$ chip updates the cache entries on the update path, and outputs the new hashes, along with a HMAC acknowledging the write. The controller updates the on-disk copy of the Merkle tree, and uses the HMAC to assure the client of the write's durability.

III. Two Chips are Cheaper than One

Straightforward approaches to securing high-performance computation that put a CPU and NVRAM in one secure package yield impractical solutions because feature size in non-volatile memory designs (currently 130nm) trails feature size in high-performance CPU designs (currently 32nm). Combining the two designs on a single die calls for a more complex manufacturing process and reduces wafer yield, resulting in unreasonably high per-unit costs.

We avoid this problem by splitting up the TCB into two chips. The ideal $\mathcal{P}$ **(processing) chip** (figure 4) is a tamper-proof FPGA with small feature size and no NVRAM, and section IV argues for the merit of this solution. For some applications, the $\mathcal{P}$ chip might be a high-speed multi-core CPU augmented with a cryptographic engine and a secure enclosure. The $\mathcal{S}$ **(state) chip** (figure 5) is a resource-constrained secure processor with NVRAM under a large feature size, such

[1] according to the BFS-traversal order, starting at 1 for the root

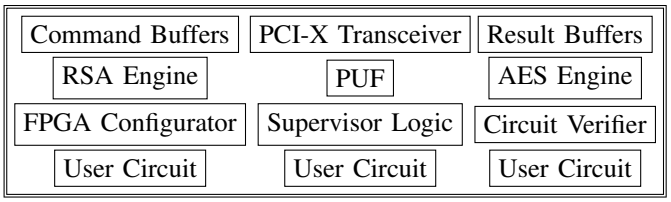

Fig. 4. The $\mathcal{P}$ chip does heavy-weight computations. It doesn't have NVRAM, and cannot maintain state across power cycles.

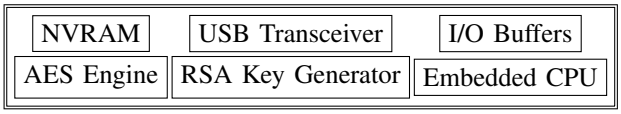

Fig. 5. The $\mathcal{S}$ chip maintains state across power cycles. It has a few kB of RAM and NVRAM, and a 2 MIPS CPU.

as a smart-card chip. The $\mathcal{S}$ chip is only responsible for holding state across power cycles, and is not on the high-performance computing path. We use a Physical Unclonable Function (PUF) [6] to bind the $\mathcal{P}$ chip to the $\mathcal{S}$ chip.

The $\mathcal{S}$ chip stores the system state across reboots, and loads it into the $\mathcal{P}$ chip when the cloud server is powered up. We achieve secure communication (guaranteed privacy and integrity) over an untrusted channel by pairing the $\mathcal{S}$ chip with the $\mathcal{P}$ chip during manufacturing. This pairing is essential to the security of the system – so far, most attacks on the TPM exploit the untrusted bus between the TPM and the CPU.

The pairing assumes a PUF on the $\mathcal{P}$ chip, which generates a symmetric encryption key. The PUF requires a syndrome (error correction information) to reliably re-generate the symmetric key. The syndrome doesn't leak information about the key [7] [8], so it can be stored in plain text, as long as the boot process checks its integrity.

The pairing is done at manufacturing time, as follows:

1) $\mathcal{P}$ chip queries its PUF to generate a symmetric key SK and a syndrome ECC.
2) $\mathcal{S}$ chip generates $(\mathsf{PrivEK}, \mathsf{PubEK})$, outputs PubEK.
3) Manufacturer signs PubEK, generates ECert, and sends it to the $\mathcal{P}$ chip.
4) $\mathcal{P}$ chip verifies the manufacturer's CA key in ECert against the key in its ROM, and outputs SK encrypted with PubEK, ECC, and its HMAC [9] $\text{HMAC}_{\mathsf{SK}}(\mathsf{ECC})$.
5) Manufacturer provides the $\mathcal{P}$ chip output to the $\mathcal{S}$ chip and stores the ECC.
6) $\mathcal{S}$ chip decrypts SK with PrivEK, stores it in NVRAM, and uses it to verify the HMAC of ECC. Upon success, it outputs a signature $\sigma_{\mathsf{PrivEK}}(\mathsf{ECC})$.
7) Manufacturer packages the $\mathcal{S}$ and $\mathcal{P}$ chips, with the public state ECert (containing PubEK), ECC, and $\sigma_{\mathsf{PrivEK}}(\mathsf{ECC})$.

The manufacturing process requires integrity guarantees in the channel between the $\mathcal{S}$ chip and the certificate issuer, similar to the TPM model. The channel between the $\mathcal{S}$ chip and the $\mathcal{P}$ chip is completely untrusted. For packaging convenience, the public state can be stored in the $\mathcal{S}$ chip's NVRAM, even though it does not require security guarantees.

When the cloud server boots, the system state is transmitted from the $\mathcal{S}$ chip to the $\mathcal{P}$ chip according to the following process:

1) Server OS presents the public state ECC, $\sigma_{\mathsf{PrivEK}}(\mathsf{ECC})$, and ECert to the $\mathcal{P}$ chip.
2) $\mathcal{P}$ chip checks the manufacturer key in ECert against the key in its ROM, and verifies ECC against $\sigma_{\mathsf{PrivEK}}(\mathsf{ECC})$ using PubEK in ECert. Upon success, ECC is fed into the PUF to recover SK.
3) $\mathcal{P}$ chip generates a boot nonce n, outputs n and its HMAC $\text{HMAC}_{\mathsf{SK}}(n)$.
4) Server OS provides the $\mathcal{P}$ chip's output to the $\mathcal{S}$ chip.
5) $\mathcal{S}$ chip verifies the HMAC, then outputs the system state s (typically a cryptographic hash), PrivEK encrypted under SK, and $\text{HMAC}_{\mathsf{SK}}(s||n)$.
6) Server OS provides the $\mathcal{S}$ chip's output to the $\mathcal{P}$ chip, together with ECert. The $\mathcal{P}$ chip checks that PrivEK corresponds to PubEK, and verifies the HMAC.
7) $\mathcal{P}$ chip decrypts PrivEK using SK, and loads the system state s in its RAM.

The above process is secure against any attack on the server software that controls the communication channel between the $\mathcal{P}$ chip and the $\mathcal{S}$ chip.

IV. Securing High-Performance Computation

In this section we argue that, given the right software infrastructure, FPGAs are better suited for secure high-performance cloud computing than secure processors such as [10] or GPUs, and we propose FPGAs as excellent candidates for $\mathcal{P}$ chips.

Today's cloud architectures use Intel x86 processors, which lets developers reuse current code and toolchains. However, general-purpose CPUs were not designed for multi-tenancy, whose security requirements clash with performance enhancements such as cache sharing. Graphical Processing Units (GPUs) are gaining adoption for cloud computing, due to the speed and power efficiency advantages that they bring to massively parallel computations. GPUs were not designed for cloud environments either, and virtualization attempts seem to be limited to time multiplexing [11]. FPGAs offer more flexibility than GPUs, and the flexibility can be used to obtain a better power-performance ratio [12]. Furthermore, FPGAs can support space-sharing, in addition to time-sharing, via partial reconfiguration using the methods in [13].

FPGAs can be used in $\mathcal{P}$ chips by following the Trusted Execution Module (TEM) [14] architecture. Computation with high security requirements or with a high degree of parallelism would be entrusted to the $\mathcal{P}$ chip. The example in section II uses the $\mathcal{P}$ chip for SHA-1 hashing, which has a very efficient FPGA implementation. The computation, expressed as a partial FPGA personality matrix, and the input data are partially encrypted with the $\mathcal{S}-\mathcal{P}$ chip pair's PubEK, which provides privacy and integrity guarantees where needed (see [14] for details). In addition to the FPGA, the $\mathcal{P}$ chip's secure enclosure contains a supervisor controller, with the same role as Xilinx's ACE controller and the TEM's firmware. The supervisor controller is connected to the off-chip bus, and

contains cryptographic engines and logic for verifying the integrity of a partially encrypted computation package, ensuring that the computation circuit specified by the personality matrix is verifiably secure, and deploying the circuit to a part of the FPGA. After the computation completes, the supervisor moves the computation's result off-chip and reclaims the FPGA resources. To simplify allocation, FPGA resources can be equally divided into a number of discrete chunks which would be atomically allocated to circuits, similarly to the way that Amazon cloud users must specify their resource usage when reserving Virtual Machines.

The main challenge in the supervisor controller, from a security standpoint, is ensuring that circuits sharing the FPGA cannot perform side-channel attacks against each other, or against the super controller in the same chip package. We propose the software approach of constraining the circuits allowed by the server to a subset than can be verified to be secure by a straightforward algorithm that does not unduly increase TCB complexity. A key observation is that, assuming two FPGA circuits are completely disjoint, the only way one of them can learn side-channel information about the other is by measuring timing variations induced by heat dissipation. Therefore, the constraints for verifiably secure circuits should forbid any constructions that may be used for such measurements. For example, a method for computing the maximum clock speed for a circuit consisting of combinational sub-circuits and registers can be used to verify that the circuit will operate without dependencies at a certain clock speed. The circuit compiler can be required to include additional information to help verification, such as a partition of the circuit into combinational sub-circuits and registers, together with timing specifications for each sub-circuit. It is helpful to notice that the area allocated to a circuit is available during the verification process, and can be used to temporarily deploy a high-speed processor, or any other construction that is useful to the verification algorithm.

V. Mitigating Side-Channel Attacks

The previous section discusses side-channel attacks that result out of multiplexing a FPGA chip. This section tackles more traditional attacks, performed outside the FPGA chip. We focus on timing and power analysis attacks, as they can be performed by malicious cloud tenants who might compromise the server hypervisor and gain access to the bus between the CPU and the FPGA, and to temperature sensors near the FPGA chip. We observe that current methods for eliminating side-channel information, surveyed below, downgrade a circuit's performance on any input to the performance of the worst-case input. We claim that it is possible to reduce the overhead of protecting against side-channel attacks, while maintaining strong security guarantees, by noticing that it is sufficient to have circuit power and time consumption be indistinguishable from the power and time consumption for random input, for a computationally bound adversary.

Power analysis became popular after the introduction of Differential Power Analysis (DPA) [15] [16], which provides more insight into a computation than timing attacks such as [17]. Sound techniques for thwarting DPA include specially designed logic gates that consume a constant amount of power per cycle [18], and decoupling the computation circuit from the power source via a circuit that hides the real power consumption [19]. "Data whitening", described in [16], is a popular heuristic approach for obfuscating the power signal by peforming most of the computations on transformed versions of the secret data (e.g., XORing the data with a random number). These techniques add overheads of 2-4x to the circuit's area and throughput.

We assume a side-channel attack model similar to [20], generalized to all kinds of side-channels. Side-channel information is leaked by consuming a resource such as time and power, or by emitting an undesirable byproduct, such as radiation, noise, or heat. We focus on resource consumption, and note that emissions can be modeled in a similar manner. We further assume that a sensitive operation consists of a sequence of not necessarily identical rounds, where each round's resource consumption depends on public input data and a subset of the secret data. The attacker's knowledge of the algorithm and implementation allows her to produce tuples of inputs that cause the circuit's consumption to vary across a set of possibilities for one or a few rounds, while the consumption across all the other rounds is identical for all the inputs in the tuple. Mapping inputs in the tuple to the specific behaviors reveals partial information about the secret, which is fed into further attacks. For example, [21] uses the fact that the modular exponentiation in RSA private-key decryption only involves a multiplication for rounds where the corresponding bit in the private exponent is set to mount a timing attack, and [22] exploits differences in the power usage of a multiplier to infer correlations between an AES key and the ciphertext.

Assuming an optimized implementation, reducing resource consumption would be infeasible, so the only method for removing information from the resource consumption signal is to spend additional resources. We explore adaptive circuits, where the amount of spent resources can be modified dynamically. Time can be adaptively spent by storing results in intermediate buffers before releasing them off-chip, and by reducing the clock speed or stoping computation altogether for short periods of time. Power can be adaptively spent by the construction in [19], by adjusting the target value for the capacitator discharge cycle. Our key insight is that a side-channel observer cannot learn any information as long as each round's resource consumption probability distribution is computationally indistinguishable from the probability distribution of running that specific round of computation on random inputs. Current side-channel defenses attempt to guarantee that resource consumption is independent of the input, which is a stronger form of the requirement above, as it essentially makes the consumption probability distribution identical for all inputs.

We propose an add-on module that interfaces between the circuit performing sensitive information and the outside world, with the ability to spend a variable amount of resources (adap-

tive power sink, output buffers and clock control). The module has a controller for coordinating resource expenditure, which is pre-loaded with average-case probability distributions for power and time consumption of the sensitive circuit, and aims to exhibit a resource consumption profile that matches that probability, multiplied by a constant that accounts for overhead. For each round, the controller invokes a random number generator (RNG) to sample the pre-programmed probability distribution, which decides the target power consumption. If the computation circuit's consumption is below the target level, the controller can accumulate resources into a deposit of slack resources, which can be depleted to make up for situations when the circuit's consumption is above the target level.

It is important to assure ourselves that an attacker cannot generate inputs that cause extreme resource consumption, such as inducing a worst-case consumption for all the rounds in the computation. Informally, an attacker that is able to put together such an input already has the secret information that the side-channel attack is intended to reveal. Attackers with no prior information on the secret can generate tuples of inputs as described above but, given a single generated input, the attacker should not be able to assert that the input will result in significantly different overall power consumption, compared to random input. Therefore, we can reasonably model per-round resource consumption as independent random variables, and reason that for a large number of rounds, their aggregate will resemble a normal distribution, which yields good bounds on the expected size of the deposit of slack resources that the controller will have to build up. This approximation only works for a large number of rounds, so the controller will exhibit high resource consumption in the first few rounds, to build up the deposits of slack resources.

VI. Conclusion

This paper introduces a new approach to building a low-cost Trusted Computing Base (TCB) for high-performance cloud computing, by pairing a chip with high processing power (the $\mathcal{P}$ chip) with a low-power secure processor with NVRAM that can carry state across power cycles (the $\mathcal{S}$ chip), such as a smartcard. We have shown a mechanism for securely pairing the chips, so that a $\mathcal{P}$ chip will always retrieve its state from the same $\mathcal{S}$ chip after booting. We argued that the ideal plaform for the $\mathcal{P}$ chip is an FPGA, as it can securely host circuits from mutually distrusting tenants. We explored a generic method for protecting the computations inside a $\mathcal{P}$ chip against side-channel attacks at lower costs than traditional methods by producing a time and power consumption signature reflecting the average-case input.

Acknowledgements

This work was supported in part by Northrop-Grumman and in part by Quanta corporation. We thank Eran Tromer and Nickolai Zeldovich for useful discussions during the course of this work.

References

[1] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 199–212.

[2] N. Zeldovich, A. Yip, F. Dabek, R. Morris, D. Mazieres, and F. Kaashoek, "Multiprocessor support for event-driven programs," in *Proceedings of the USENIX 2003 Annual Technical Conference*, 2003, pp. 239–252.

[3] Trusted Computing Group, "Trusted Platform Module (TPM) Specifications," https://www.trustedcomputinggroup.org/specs/TPM/.

[4] J. Li, M. Krohn, D. Mazières, and D. Shasha, "Secure untrusted data repository (SUNDR)," *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation-Volume 6 table of contents*, pp. 9–9, 2004.

[5] R. Merkle, "A certified digital signature," in *manuscript*, 1979.

[6] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proceedings of the 44th Conference on Design Automation*, 2007.

[7] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: how to generate strong keys from biometrics and other noisy data," in *Advances in Cryptology - Eurocrypt 2004*, 2004.

[8] M.-D. M. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Design and Test of Computers*, vol. 27, pp. 48–65, 2010.

[9] H. Krawczyk, M. Bellare, and R. Canetti, "RFC 2104: HMAC: Keyed-Hashing for Message Authentication," Feb. 1997.

[10] G. Suh, C. O'Donnell, and S. Devadas, "AEGIS: A single-chip secure processor," *Information Security Technical Report*, vol. 10, no. 2, pp. 63–73, 2005.

[11] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan, "Gvim: Gpu-accelerated virtual machines," in *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*. ACM, 2009, pp. 17–24.

[12] S. Che, J. Li, J. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs," in *IEEE Symposium on Application-Specific Processors*. IEEE, 2008.

[13] O. Diessel, H. ElGindy, M. Middendorf, H. Schmeck, and B. Schmidt, "Dynamic scheduling of tasks on partially reconfigurable FPGAs," in *Computers and Digital Techniques, IEE Proceedings-*, vol. 147, no. 3. IET, 2000, pp. 181–188.

[14] V. Costan, L. F. G. Sarmenta, M. van Dijk, and S. Devadas, "The trusted execution module: Commodity general-purpose trusted computing," in *CARDIS*, 2008.

[15] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in CryptologyCRYPTO99*. Springer, 1999, pp. 789–789.

[16] T. Messerges, "Using second-order power analysis to attack DPA resistant software," in *Cryptographic Hardware and Embedded SystemsCHES 2000*. Springer, 2000, pp. 27–78.

[17] P. Kocher, "Timing Attacks on Diffie-Helman, RSA, DSS and Other Systems," in *Proceedings of Advances in Cryptology–Crypto*, vol. 96, 1996, p. 104.

[18] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Proceedings of the conference on Design, automation and test in Europe-Volume 1*. IEEE Computer Society, 2004, p. 10246.

[19] C. Tokunaga and D. Blaauw, "Securing Encryption Systems With a Switched Capacitor Current Equalizer," *Solid-State Circuits, IEEE Journal of*, vol. 45, no. 1, pp. 23–31, 2010.

[20] M. Joye, P. Paillier, and B. Schoenmakers, "On second-order differential power analysis," *Cryptographic Hardware and Embedded Systems–CHES 2005*, pp. 293–308, 2005.

[21] D. Brumley and D. Boneh, "Remote timing attacks are practical," in *Proceedings of the 12th conference on USENIX Security Symposium-Volume 12*. USENIX Association, 2003, pp. 1–1.

[22] S. Ors, F. Gurkaynak, E. Oswald, and B. Preneel, "Power-Analysis Attack on an ASIC AES implementation," in *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, vol. 2. IEEE, 2004, pp. 546–552.