Tiny Trusted Third Parties*

Alexander Iliev
sasho@cs.dartmouth.edu

Sean Smith
sws@cs.dartmouth.edu

April 18, 2005

Abstract

Many security protocols hypothesize the existence of a *trusted third party (TTP)* to ease handling of computation and data too sensitive for the other parties involved. Subsequent discussion usually dismisses these protocols as hypothetical or impractical, since trusted third parties clearly cannot exist. However, we believe this assumption is fallacious: the last decade has seen the emergence of hardware-based devices that, to high assurance, can carry out computation unmolested; emerging research promises more. Unfortunately, perhaps as an inevitable consequence of cost-effective physical security, these devices have small memory and limited computational power.

In theory, such devices can perform the role of a trusted third party in real-world problems. In practice, we have found that these devices are too small to be applied naively as TTPs for large problems. However, we have found that we can often build practical solutions anyway, using collections of these tiny TTPs carefully integrated. This thinking suggests a new paradigm in thinking about algorithms, hardware, and complexity.

1 Introduction

Computer security protocols typically consist of computation distributed between two or more parties, potentially adversarial and with different interests and motivations. To ensure that this computation possesses the desired security and privacy properties (balancing the interests of these participants), designers often posit the existence of a *trusted third party (TTP)*. If the main parties trust this TTP not to cheat, then properties and results that might otherwise require delicate cryptographic interaction between the main parties can be obtained simply by moving the right computation to the TTP. Often, the addition of a TTP adds sufficient flexibility and efficiency to turn the protocol from impractical to feasible for the real world.

However, many regard the use of TTPs in protocols itself as cheating, on the part of the designer. What organization or humans are sufficiently trustworthy? Even if an organization might be sufficiently trustworthy right now, what about the future? Consequently, although the addition of a TTP may make a design feasible from a computational perspective, it becomes akin to using magic [25] or fairies: a technical approach not particularly appropriate for the real world.

Secure Multiparty Computation Theoretical computer science has developed the tool of *secure multiparty computation (SMC)* to address this problem—in theory. SMC enables entities to collaborate with others while preserving the privacy of information they would rather not expose.

The notion of a trusted third party is an important one for SMC. It serves to define what it means for a multi-party protocol to be secure. The ideally secure protocol is one where all the players send their inputs to a universally trusted party who computes the specified function, and then sends each participant their intended result. Thus all the participants only

*Review submission, NSPW 2005

learn their result and nothing more. In practice, most work on SMC declares that actually using a TTP is not reasonable, as there are no real trusted parties. Instead, cryptographic protocols are developed which achieve a similar security, but involve only the participants and no third parties. The overhead of such protocols is usually quite high however.

Recently, Malkhi et al. have offered a preliminary prototype [19], which however has high overhead in some circumstances, such as programs which access their (vector) inputs in an input-dependent manner.

Hardware On the other hand, computing devices designed to be trustworthy for relying parties at remote locations have been designed and built. Such devices do actually offer a possible incarnation of a TTP which does not rely on organizational trust, but only on technological provisions, like strong physical security and self-authentication mechanisms.

In this paper, we focus on such trustworthy devices and argue, on the basis of our experience building them (e.g., [26, 27, 20]) and using them in real applications, that they are ready to evolve to a form which may be able to support private computations at scales interesting on a practical level.

Such devices were also investigated, though only theoretically, in the series of articles 10-15 years ago by Goldreich and Ostrovsky on Oblivious RAMs [7, 9].

Tiny TTPs Unfortunately, these trustworthy devices tend to be small in computational power and in memory. These limitations may even be inevitable; a TTP armored against an adversary with physical access might not have a large memory because of the difficulty of armoring a large device, nor run at high speed due to the difficulty of dissipating heat.

We have nonetheless designed and implemented such TTP algorithms working around these low space requirements. However, we have found that a certain kind of stream processing is by far the most time-consuming task performed by a TTP in many SMC applications.

Thus, we suggest that the evolution of the current generation of TTPs should be a *Tiny* TTP, with a small memory size, small processing units optimized for fast stream processing, and very fast IO channels. Such a device should be able to serve as a practical TTP for efficient secure multi-party protocols on interesting dataset sizes. We also suggest that effective use of these tiny TTPs requires a new way of thinking about "secure" computation.

This Paper This paper develops our vision for using tiny TTPs to build practical solutions to real security problems. Section 2 presents some background in hardware-based TTPs, and Section 3 presents some of our experience using tiny TTPs for big problems. We then discuss some of the new paradigms that a systematic exploration of tiny TTPs would entail: Section 4 projects implications for hardware architecture, Section 5 projects implications for algorithm design, and Section 6 projects implications for complexity analysis. Section 7 discusses related work.

2 Background

We start by considering some background.

Organizational TTPs Traditionally, researchers tend to imagine a TTP as some organization-based entity. In many applications, a third party can offer trust mostly on the basis of being in a different, presumably impartial, organization than the participants in the computation. Certification authorities usually fall into this category—a CA is trusted because, as an organization, it is impartial about the secure computations its certificates protect, and it must remain honest in order to maintain its reputation. However, there is no technological way in which a relying party can decide if a given certificate from a CA was actually generated properly.

This is presumably the kind of TTP that most people would not want to mediate their multiparty secure computations.

Secure coprocessors However there do exist devices which can offer assurance that they are performing a task as expected, independent of how trustwor-

thy their operator is. This is the programmable *secure coprocessor* (SCop). A secure coprocessor is a small general purpose computer armored to be secure against physical attack, such that code running on it has some assurance of running unmolested and unobserved [31]. It also includes mechanisms (*outbound authentication*, also called *attestation*) to prove that some given output came from a genuine instance of some given code running in an untampered coprocessor [24]. The coprocessor is attached to a *host* computer. The SCop is assumed to be trusted by clients (by virtue of all the above provisions), but the host is not trusted (not even its root user). The strongest adversary against SCop applications is the superuser on the host (perhaps also armored with a drill).

In the late 1990s, a programmable SCop emerged as a commercial device. Now in its third generation, the IBM 4758 [27] was validated to the highest level of software and physical security scrutiny[1] offered then—FIPS 140-1 level 4 [29]. It has an Intel 486 processor at 99 MHz, 4MB of RAM and 4MB of FLASH memory. It also has cryptographic acceleration hardware combined with FIFOs, enabling fast streaming and decryption or decryption of data into or out of the module [18]. It connects to its host via a PCI bus. In production, the 4758 runs the CP/Q++ embedded OS; however, experimental research devices can run a version of Linux (as does the follow-on product from IBM, the PCIXCC [1]).

More Limited TTPs The 4758 and its follow-ons are too large to be portable and too expensive to be ubiquitous. However, other emerging research (e.g., [30, 17]) and even commercial efforts (e.g., [10, 28]) explore a different approach: shrinking the physical security boundary to a single chip, such as a portion of the CPU. This approach can potentially decrease cost and increase ubiquity—at the expense of making the TTP even tinier (e.g., limited by the amount of code and data that can fit into an internal cache).

[1]Since its release, a few groups have found flaws in specific application software—but the fundamental platform has survived unscathed.

3 Experience

We have made extensive use of SCops in implementing a practical *Private Information Retrieval* system, which is an important instance of SMC.

All the solutions we have studied and implemented share some structure, which can be used to design more efficient SCop architectures, much like graphics processing benefits from optimized hardware to deal with a small set of computationally intensive primitives. In this section we outline the structure of these solutions.

3.1 Private Information Retrieval

The PIR problem is: Alice has a database of items $X = \langle x_1, x_2, \ldots, x_N \rangle$, Bob has an index $i \in [1..N]$, Alice should learn x_i without Bob learning i.

Our work focuses on providing practical solutions with the assistance of a SCop [11, 12]. We assume that the SCop is small: if the items are of size $\mathrm{O}(M)$, that the SCop can only fit $\mathrm{O}(M + \log N)$ bits, which is the minimum required to fit a constant number of items and indices. The whole database, and any additional working storage, resides on the SCops host and is fully accessible to the adversary.

The basic algorithm we use was initially developed by Goldreich and Ostrovsky in their *oblivious RAM* work [9], and later also used by Asonov [2]. The outline is that the SCop generates a randomly and *obliviously* permuted (and of course encrypted) version of the database, $\pi(X)$. The host sees a sequence of ciphertexts. "Oblivious" means that the permutation π is completely hidden from the host—as are the keys necessary for decryption and integrity-checking. Then, the SCop can service a query for item i from Bob just by reading in one item, $\pi(X)_{\pi(i)}$ from the permuted database. Since the host does not know π, he cannot learn i. Each permuted database can only be used a limited number of times before a new one, using a new random permutation, needs to be generated.[2]

[2]This is a simplified exposition, we refer the reader to [11, 12] for the full details—as well as how we reduce the internal space bound.

The most time-consuming part of this scheme is permuting the database obliviously while using only very little trusted storage. It can be achieved in $O(N \log^2 N)$ time using Batcher's bitonic sorting network [3]. Such networks are an important theme here–we elaborate on them in the next section.

Since the permutation step is the most time consuming, it makes sense to try to optimize it, as we discuss later.

To give an idea of the performance of our prototype, with a database of

3.2 Oblivious Networks

A common theme in our work was the need for the SCop to perform operations *obliviously* on a sequence S of N items residing on the host. Operations include permuting with a given permutation π, sorting with a given sorting key $k(s_i)$, and merging two sorted subsequences. In all cases the obliviousness requirement is that the adversary (bounded to feasible computations) does not learn anything about the parameters of the operation from observing the IO that the SCop performs.

From other fields of computer science comes the construct of *networks* which can perform any of these tasks. These networks consists of small operators whose settings are operation-dependent (eg. depend on the permutation π), wired together in a fixed manner, independent of the operation.

Example: Permutation network A Beneš permutation network can perform any permutation π of N input items by passing them through $O(N \lg N)$ crossbar switches (the operators), each of which operates on two items, either crossing them or passing them straight. The crossbar settings differ for different π, but the connections between the switches is fixed for a given input size N. In Figure 1 we illustrate a small Beneš network.

A bitonic sorting network is similar to the Beneš, but it sorts its input, and consists of $\frac{N}{4} \log^2 N$ *comparators*, each of which sort two items. The comparators are arranged in $\frac{\log^2 N}{2}$ stages.

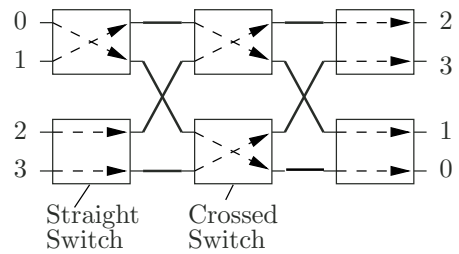

Figure 1: A Beneš permutation network with 4 inputs, performing the permutation $\langle 2, 3, 1, 0 \rangle$. The dashed lines represent switch settings, which depend on the permutation. The rest of the network only depends on the number of inputs.

These networks are useful for our problem because (1) the SCop can use cryptography to perform a single operator on two items resident on the host without the host learning what the operator did, and (2) by doing this for all the operators in the network, the SCop can process the whole dataset without the host learning anything about the operation, even though he observes all the item I/O.

More specifically, in the Beneš network case, to execute a switch the SCop reads in the two items involved, internally crosses them or not, and writes them out encrypted under a new key so the host cannot tell if it was a cross or not. Since the network consists of $2 \lg N$ stages of switches with $N/2$ switches each, and the SCop can execute the switches stage by stage, he can use one key per stage, thus never needing to store more than two keys at a time during the permute operation.

4 New Paradigms: Hardware

For all the PIR and PIRW prototypes we built, the main bottleneck is streaming the whole dataset $O(\log N)$ or $O(\log^2 N)$ times through the SCop, once for each stage of the permutation or sorting network being performed. The computation required to set up the network, which is $O(N \log N)$ time for the

Beneš[3], but does not involve much I/O, was negligible in comparison to the time needed to perform the network.

Looking at our experience with the 4758 secure coprocessor, we suggest a re-design of the device to take advantage of the regular nature of the most time-consuming tasks it has to perform: streaming the dataset through the SCop while performing simple operations on adjacent items, and re-encrypting the items. The remaining internal computation can be fairly weak—and indeed, a custom-designed 2-in, 2-out crypto switch might even reduce the needed internal memory to a constant number of blocks of the underlying block cipher, rather than whole database items.

We propose two new design principles—making the main memory and controller of the TTP small, and adding an optimized unit with a small local RAM to deal with stream-processing of the dataset.

The 4758 SCop already has a specialized component for a streaming operation, which is the DES engine. It can be configured with a key, and set to stream data from the outside, through a DES operation and back out, without touching the main CPU or RAM.

We propose to expand this design with a *streaming processing unit* (SPU), which is similar to the 4758 DES engine, and the newer Otello cryptographic engine in the PCIXCC, in that it is optimized to handle a stream of data and is configurable in a "fast path" to and from the outside, but it will be programmable.

Also we propose to generalize the 4758 and PCIXCC notion of a cryptographic engine, and make that component programmable too. Recent advances in processor design for symmetric cryptography should make it efficient to program block ciphers in software, as opposed to relying on hardware acceleration as is the standard for high-performance cryptography. In particular, one of the main tools important for implementing block ciphers but missing from general purpose instruction sets is a general bit-permutation instruction. Recent work has examined how such instructions can be provided efficiently [16].

[3]Setting up the Beneš network consists of computing all the switch settings for the permutation π, which can be done in $O(N \log N)$ time

The advantages of having a programmable cryptographic processor are many. One is extra flexibility in using the engine, for example being able to configure it to only process parts of each block of data, and leave other parts in cleartext. In the SCop scenario, data intended to be read by the host should be left in the clear. Another advantage is the ability to install new block ciphers and modes (such as the ability to verify integrity and decrypt together, in one pass), seeing that this area of cryptographic research is far from static.

One proposed design of the device in outline is shown in Figure 2.

4.1 An example SPU program

We demonstrate how the SPU could be programmed in C in Figure 3. The SPU should in general not be concerned with encrypting and decrypting its data stream. It may have a hardware-optimized `swap` functionality, as this is frequently used in oblivious networks.

4.2 Relation to Cell processor

Recent Commercial efforts have produced another radically new special-purpose CPU with a similar structure to our proposed design—the Cell processor, targeted at game consoles (the Sony playstation 3). This consists of a POWER processor element (PPE), multiple synergistic processor elements (SPE), and a high-speed element interconnect bus (EIB) [23, 14]. The SPE's each have a small local memory, which is not coherent with the PPE's memory, and they are intended to deal with processing of streaming data. The graphics-intensive applications to which the Cell processor is targeted also share some structure with oblivious network processing—local processing of large volumes of streaming data.

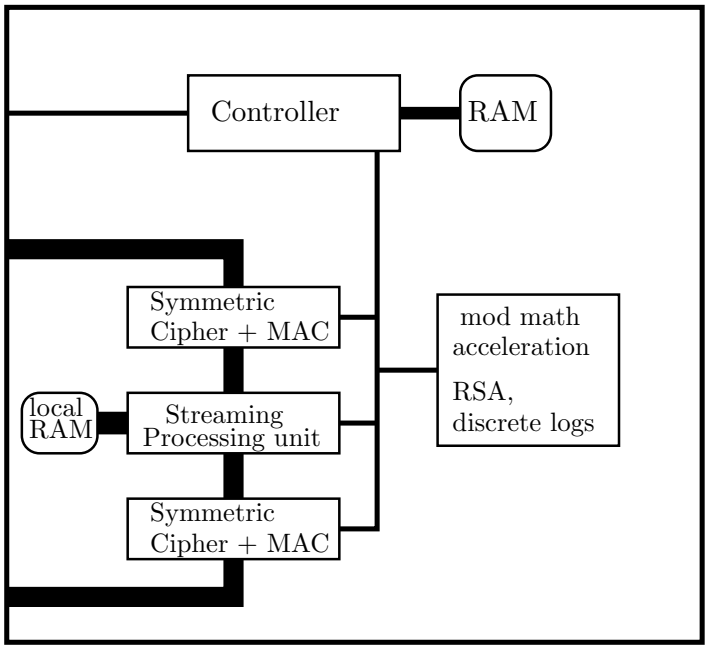

Figure 2: Design of a stream-processing optimized Tiny TTP. The interconnect thickness is proportional to bandwidth.

5 New Paradigms: Algorithms

5.1 SMC using blinded circuits with RAM

We are currently experimenting with enhancing two-party secure computation, using a tiny TTP to provide RAM to the standard blinded circuit solution.

General solutions to the secure two-party computation problem usually make use of a circuit representation of the function f to be computed, such that one player (say Bob) "blinds" the circuit and the other (Alice) executes the blinded circuit, without being able to interpret the values; see eg. [8].

Blinding here means that Bob replaces the usual bit values (0 and 1) with different random keys for each wire in the circuit, and replaces the circuit gates with tables which allow the propagation of these keys in accordance with the semantics of the gates. Thus, given the right keys at the input wires, Alice can compute the corresponding keys for the output wires, without knowing which bit value any of the keys correspond to.

Recently an actual implementation, Fairplay, of a general secure two-party computation system has been constructed [19]. Some functions however impose a large overhead to be represented as a circuit, in particular if they make use of arrays indexed in an input-dependent manner[4].

We are experimenting with using TTPs to provide a RAM for circuit-based two-party computation, as an extension to the Fairplay system. This RAM will reside on an untrusted host, but be accessed obliviously by a Tiny TTP residing on that host, using similar mechanisms as our previous PIR work.

[4] In general circuits impose a high overhead in simulating indirect indexing, which is the hallmark of RAM machines.

```
size_t ITEMSIZE;

void * temp = alloc (ITEMSIZE);

/* structure to keep track of the expected comparator, so we catch any
   substitution or replay attacks */
struct comparator_addr {
    int stage;                  /* which stage of the network */
    int idx1, idx2;             /* the two item indices */
};


comparator_addr current_addr;


/* called by the controller at the start of every stage */
void stage_init (int stage_num) {
    current_addr.stage = stage_num;
}

/* called whenever a whole block is available */
void * process (void * block) {

    void * a = block;
    void * b = block + ITEMSIZE;

    update_addr (&current_addr); /* advance current_addr to the next comparator
                                  * (ie. this one) */

    /* make sure it's the expected block */
    if ( stage (a) != current_addr.stage || index(a) != current_addr.idx1 || ... ) {
        fail ("Incorrect items for comparator");
    }

    /* the actual comparator */
    if (key(a) < key(b)) {
        swap (a, b, temp);      /* using 'temp' for the swap */
    }
    else {
        swap (a, a, swap);      /* null swap to keep timing the same */
    }

    return a;                   /* done with this block, indicate it can be passed on */
}
```

Figure 3: Sample SPU program. The `process` function is a callback called whenever a whole block has been input. The block size is set by the controller in advance. `stage_init` is called at the start of every stage of the network. Also at the start of a stage, the controller will configure the crypto engines to deal with decrypting incoming data and decrypt outgoing data. Then the controller will start the flow of data into and out of this path.

5.2 Private advertising

Online newspapers sometimes expect their users to register and log in every time they read the paper. Ostensibly the reason is to enable targeted delivery of advertisements, based on a long-term record of the person's reading habits. Such a dossier is of course a potential privacy risk, especially if it is exposed to external parties, through a compromised server (or laptop!) for example.

We have thought about applying PIR to this problem—enabling a user to retrieve web pages privately, but enable the TTP to track the person's selections and compile ads accordingly. The dossier would in this case be guaranteed to be used only as intended, and no-one would learn the user's interests.

The New York Times has about 25 sections, with up to 20 articles in each section, and perhaps 5 extra objects (images, etc.) specific to each article, giving about 2500 objects in total. Some of these may be large and need to be split, giving perhaps 5000 items in the dataset to be privately served. This dataset size is within the reach of our current PIR prototype, but would have an average response time of about 8 seconds, per item, which could clearly be excessive considering that multiple items will make up one page. Thus, a considerably faster implementation (ie. on a faster device) is needed to enable this application.

5.3 Beyond

More generally, this work has already made practical use of previously "impractical" oblivious RAM work and largely forgotten Beneš networks, to produce an arguably practical solution to PIR, and potentially to increase the feasibility of SMC. What else can be achieved, if we continue to mine the "impractical" literature?

6 New Paradigms: Complexity

A major theme in the use of Tiny TTPs is that they should compute obliviously. Informally this means that even though the TTP may be using a large data structure (encrypted) on untrusted storage, the access pattern to the structure should look the same to an adversary no matter what the actual data in the structure is. Currently there exist general solutions for simulating a RAM program obliviously–the Oblivious RAMs work by Goldreich and Ostrovsky [9], but with considerable overhead—either a factor of $O(\log^4 N)$ with high constants, or a factor of $O(\sqrt{N} \log N)$ with lower constants.

Thus, an obvious question is whether some specific problems can be solved obliviously with less overhead over their non-oblivious solution. We have already encountered permuting and sorting, where obliviousness imposes a factor of $O(\log^2 N)$ and $O(\log N)$ respectively (using the bitonic sorting network in both cases).

An interesting question is to investigate the class of problems which are solved by oblivious algorithms with small overhead. One example of an algorithm which is oblivious by nature is the Bellman-Ford graph shortest path algorithm. It makes a fixed number of complete passes over the edges of the graph, so its access pattern is the same (for a given number of vertices and edges) regardless of the actual graph.

Another interesting question is what is the appropriately general model for "things that can be done securely with a network of tiny TTPs." If we move beyond mere encrypted switches, what else can we do? What other problems lie in this class?

7 Related Work

Cryptographically weak devices *Remotely keyed encryption* schemes seek to enable high-bandwidth encryption (on a host machine) using long-term keys held in low-bandwidth devices like smart cards [5]. This work shares the theme of enabling large computations using a small trusted space, but is otherwise quite different as it has no obliviousness requirements, and an adversary controlling the host can decrypt ciphertext until he is removed.

In a similar space, Modadugu et al. have developed a prototype using an untrusted host to help a Palm Pilot with the computation of generating RSA

keys [22].

Secure hardware Research teams at Princeton (e.g., [21]) and MIT (e.g. [30]) have continued to explore building CPU-level versions of secure coprocessors. An ongoing project at Princeton, NPS, and USC-ISI explores re-engineering CPU architecture from the beginning, with security as a principle [15].

Less trusted and less functional TTPs The construct of *commodity servers* was introduced by Beaver to provide a trade-off between a heavyweight secure computation without *any* third parties, and a computation which fully trusts a third party [4]. Commodity servers assist a computation by providing random but potentially structured input to the players. The servers do not learn the player's actual inputs.

A similar scheme for PIR protocols appears in [6].

Private advertising Juels describes schemes which seek to combine weaker versions of PIR with targeted advertising [13]. One of the schemes actually provides anonymization (ie. hiding the user's identity from the server) rather than PIR (which hides the identity of the item retrieved), and both schemes assume that the user selects the ad to view based on his (privately-held) profile, and a selection function sent by the server.

TCG The *Trusted Computing Group* (formerly known as the *Trusted Computing Platform Association*) has specified an architecture that takes a different approach: shrinking the physical security boundary to a single chip (the *Trusted Platform Module, TPM*) and then integrating this chip into the boot process of a larger, exposed system.

References

[1] T. Arnold and L. van Doorn. The IBM PCIXCC: A new cryptographic coprocessor for the IBM eServer. *IBM Journal of Research and Development*, 48:475–487, May 2004.

[2] D. Asnonov. *Querying Databases Privately: A New Approach to Private Information Retrieval*. Springer-Verlag LNCS 3128, 2004.

[3] Kenneth Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computer Conference*, volume 32, pages 307–314, Atlantic City, NJ, USA, April 1968. Thomson Book Company.

[4] Donald Beaver. Server-assisted cryptography. In *NSPW '98, Proceedings of the 1998 Workshop on New Security Paradigms*, Charlottsville, VA, USA, September 1998. ACM.

[5] Matt Blaze, Joan Feigenbaum, and Moni Naor. A formal treatment of remotely keyed encryption (extended abstract). In *Advances in Cryptology - EUROCRYPT '98: International Conference on the Theory and Application of Cryptographic Techniques*, volume 1403 of *LNCS*, pages 251–265, Espoo, Finland, May 1998. Springer-Verlag.

[6] Yael Gertner, Shafi Goldwasser, and Tal Malkin. A random server model for private information retrieval, or how to achieve information theoretic PIR avoiding database replication. In *RANDOM '98: Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 200–217, London, UK, 1998. Springer-Verlag.

[7] O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 182–194. ACM Press, 1987.

[8] Oded Goldreich. *Foundations of Cryptography Volume 2: Basic Applications*, chapter 7: General Cryptographic Protocols, pages 599–764. Cambridge University Press, May 2004.

[9] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[10] T. Halfhill. ARM Dons Armor: TrustZone Security Extensions Strengthen ARMv6 Architecture. Microprocessor Report 8/25/03-01, August 2003.

[11] Alexander Iliev and Sean Smith. Private information storage with logarithmic-space secure hardware. In *3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems*, pages 201–216, Toulouse, France, August 2004. IFIP, Kluwer Academic Publishers. Workshop under 19th IFIP International Information Security Conference–SEC2004.

[12] Alexander Iliev and Sean Smith. Protecting client privacy with trusted computing at the server: Two case studies. *IEEE Security and Privacy*, 3(2), March 2005.

[13] Ari Juels. Targeted advertising ... and privacy too. In *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*, pages 408–424, London, UK, 2001. Springer-Verlag.

[14] Kevin Krewell. Cell moves into the limelight. Microprocessor Report, February 2005.

[15] R. Lee, C. Irvine, and T. Benzel. Research Agenda for Unified Core Mechanisms in Highly Secure Mobile Platforms. In *Security Challenges at the Foundation: Secure Computing Enabled by Hardware, Firmware and Low-Level Software*. DARPA Invitational Workshop, 2004.

[16] R. B. Lee, R. L. Rivest, M. J. B. Robshaw, Z. J. Shi, and Y. L. Yin. Permutation operations in block ciphers. In Nadia Nedjah and Luiza de Macedo Mourelle, editors, *Embedded Cryptographic Hardware: Design and Security*, chapter 13. Nova Science Publishers, Hauppauge, NY, USA, 2004.

[17] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectural Support for Copy and Tamper Resistant Software. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 168–177, November 2000.

[18] M. Lindemann and S.W. Smith. Improving DES Coprocessor Throughput for Short Operations. In *Proceedings of the 10th USENIX Security Symposium*, pages 67–81, August 2001. `http://www.cs.dartmouth.edu/~sws/papers/des.pdf`.

[19] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay—a secure two-party computation system. In Matt Blaze, editor, *13th USENIX Security Symposium*, pages 287–302. USENIX, August 2004.

[20] J. Marchesini, S.W. Smith, O. Wild, J. Stabiner, and A. Barsamian. Open-Source Applications of TCPA Hardware. In *20th Annual Computer Security Applications Conference*. IEEE Computer Society, December 2004. `http://www.cs.dartmouth.edu/~sws/papers/acsac04.pdf`.

[21] John P. McGregor and Ruby B. Lee. Protecting Cryptographic Keys and Computations via Virtual Secure Coprocessing. In *11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Workshop on Architectural Support for Security and Anti-virus (WASSA)*, October 2004.

[22] N. Modadugu, D. Boneh, and M. Kim. Generating rsa keys on the palmpilot with the help of an untrusted server. In *RSA Data Security Conference and Expo*, 2000.

[23] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa. The design and implementation of a first-generation cell processor. In *2005 IEEE International Solid-State Circuits Conference*, pages 184–186. IEEE, 2005.

[24] Sean Smith. Outbound authentication for programmable secure coprocessors. In *7th European Symposium on Research in Computer Science*, October 2002.

[25] S.W. Smith. Magic Boxes and Boots: Security in Hardware. *IEEE Computer*, 37(10):106–109, October 2004. `http://www.cs.dartmouth.edu/~sws/papers/magic_boxes.pdf`.

[26] S.W. Smith. *Trusted Computing Platforms: Design and Applications*. Springer, 2005. `http://www.springeronline.com/sgw/cda/frontpage/0,11855,4-148-22-365207%21-0,00.html`.

[27] S.W. Smith and S. Weingart. Building a High-Performance, Programmable Secure Coprocessor. *Computer Networks*, 31:831–860, April 1999. `http://www.cs.dartmouth.edu/~sws/papers/sw.pdf`.

[28] N. Stam. Inside Intel's Secretive 'LaGrande' Project. `http://www.extremetech.com/`, September 2003.

[29] National Institute Of Standards and Technology. Security requirements for cryptographic modules. `http://csrc.nist.gov/publications/fips/fips140-1/fips1401.htm`, Jan 1994. FIPS PUB 140-1.

[30] G. Edward Suh, Charles W. O'Donnell, Ishan Sachdev, and Srinivas Devadas. Design and implementation of the aegis single-chip secure processor using physical random functions. In *Proc. of the 32nd Annual International Symposium on Computer Architecture*, Madison, WI, USA, June 2005.

[31] Bennet S. Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, 1994.