RSA Laboratories

Volume 6, No. 2, Summer 2003

CryptoBytes

CONTENTS

I. A Survey of Two Signature Aggregation Techniques

Dan Boneh, Craig Gentry, Ben Lynn and Hovav Shacham

ABSTRACT

We survey two recent signature constructions that support signature aggregation: Given n signatures on n distinct messages from n distinct users, it is possible to aggregate all these signatures into a single signature. Aggregate signatures are useful for reducing the size of certificate chains (by aggregating all signatures in the chain) and for reducing message size in secure routing protocols such as SBGP.

II. On the Cost of Factoring RSA-1024

Adi Shamir and Eran Tromer

ABSTRACT

Recent work on factorization has placed the bottleneck at the sieving step of the Number Field Sieve algorithm. We present a new implementation of this step, based on a proposed custom-built hardware device that achieves a very high level of parallelism "for free" by use of algorithms that take advantage of certain trade-offs in chip manufacturing technology. Using this hypothetical device (and ignoring the initial R&D costs), it appears possible to break a 1024-bit RSA key in one year using a device whose cost is about $10M.

EDITOR'S NOTE: For RSA Laboratories' comments on the TWIRL design and our current recommendations on RSA key size, please see

http://www.rsasecurity.com/rsalabs/technotes/twirl.html

III. Physical One-Way Functions

Ravikanth S. Pappu

ABSTRACT

Physical One-Way Functions (POWFs) provide a novel approach to assigning unique, tamper-resistant and unforgeable identifiers to everyday objects. POWFs can be obtained from the inherent three-dimensional microstructure of a large class of physical systems known as mesoscopic systems, which are inexpensive to fabricate and prohibitively difficult to duplicate. In this paper, we describe implementations of, attacks on and applications of POWFs.

RSA LABORATORIES

I. A Survey of Two Signature Aggregation Techniques

Dan Boneh
dabo@cs.stanford.edu

Craig Gentry
cgentry@docomolabs-usa.com

Ben Lynn
blynn@cs.stanford.edu

Hovav Shacham
hovav@cs.stanford.edu

Abstract

We survey two recent signature constructions that support signature aggregation: Given n signatures on n distinct messages from n distinct users, it is possible to aggregate all these signatures into a single signature. This single signature (and all n original messages) will convince any verifier that the n users signed the n original messages (i.e., for $i = 1, \ldots, n$ user i signed message number i). We survey two constructions. The first is based on the short signature scheme of Boneh, Lynn, and Shacham and supports general aggregation. The second, based on a multisignature scheme of Micali, Ohta, and Reyzin, is built from any trapdoor permutation but only supports sequential aggregation. Aggregate signatures are useful for reducing the size of certificate chains (by aggregating all signatures in the chain) and for reducing message size in secure routing protocols such as SBGP.

1 Introduction

Security systems often manage signatures on many different messages generated by many different users. For example, in a Public Key Infrastructure (PKI) of depth n, user signatures are accompanied by a chain of n certificates. The chain contains n signatures by n Certificate Authorities (CAs) on n distinct certificates. Similarly, in the Secure BGP protocol (SBGP) [16] each router receives a list of n signatures attesting to a certain path of length n in the network. A router signs its own segment in the path and forwards the resulting list of $n+1$ signatures to the next router. As a result, the number of signatures in routing messages is linear in the length of the path. Both systems would benefit from a method for compressing the list of signatures on distinct messages issued by distinct parties. For example, certificate chains could be shortened by compressing the n signatures in the chain into a single signature. Note that one would still need to store the data in all certificates in the chain — only the signatures in the chain are compressed.

An aggregate signature scheme enables us to achieve precisely this type of compression. In this paper we survey two mechanisms for signature aggregation: general aggregation and sequential aggregation. We assume each of n users has a public-private key pair (PK_i, SK_i). User i wishes to sign message M_i.

General aggregate signatures. In a general signature aggregation scheme each user i signs her message M_i to obtain a signature σ_i. Then anyone can use a public aggregation algorithm to take all n signatures $\sigma_1, \ldots, \sigma_n$ and compress them into a single signature σ. Moreover, the aggregation can be performed incrementally — signatures σ_1, σ_2 can be aggregated

into σ_{12} which can then be further aggregated with σ_3 to obtain σ_{123}, and so on. There is also an aggregate verification algorithm that takes $PK_1,\ldots,PK_n$, $M_1,\ldots,M_n$, and σ and decides whether the aggregate signature is valid. Thus, an aggregate signature provides non-repudiation at once on many different messages by many users. We refer to this mechanism as *general* aggregation since aggregation can be done by anyone and without the cooperation of the signers. In the next section we describe a general aggregate signature scheme due to Boneh, Gentry, Lynn, and Shacham [5]. The scheme uses bilinear maps from algebraic geometry.

Sequential aggregate signatures. In a sequential aggregation scheme, signature aggregation can only be done during the signing process. Each signer in turn sequentially adds her signature to the current aggregate. Thus, there is an explicit order imposed on the aggregate signature and the signers must communicate with each other during the aggregation process. Operationally, sequential aggregation works as follows: User 1 signs M_1 to obtain σ_1; user 2 then combines σ_1 and M_2 to obtain σ_2; and so on. The final signature σ_n binds user i to M_i for all $i = 1,\ldots,n$. In Section 3 we describe a sequential aggregate signature scheme based on homomorphic trapdoor permutations such as RSA. The scheme is based on a multisignature scheme due to Micali, Ohta, and Reyzin [19] and analyzed in [27].

Although general aggregation is more powerful than sequential aggregation, the fact that sequential aggregation can be built from standard primitives such as RSA has its benefits. Interestingly, either mechanism can be used for compressing signatures in a certificate chain.

Aggregate signatures are related to multisignatures [24, 23, 20, 3]. In multisignatures, a set of users all sign the *same message* and the result is a single signature. Recently, Micali, Ohta, and Reyzin [20], presented a clear security model and new constructions for multisignatures. Another efficient construction was presented by Boldyreva [3]. Multisignatures are insufficient for the applications we have in mind, such as certificate chains and SBGP. For these applications we must be able to combine signatures on distinct messages into an aggregate.

The application of aggregate signatures to compressing certificate chains is related to an open problem posed by Micali and Rivest [21]: Given a certificate chain and some special additional signatures, can intermediate links in the chain be cut out? Aggregate signatures allow the compression of certificate chains without any additional signatures, but a verifier must still be aware of all intermediate links in the chain.

2 General Aggregate Signatures

In a general aggregate signature scheme, signatures are generated by individual users. They can then be combined into an aggregate signature by some aggregating party. The aggregating party need not be one of the users, and need not be trusted by them. Every aggregate signature scheme is a generalization of an ordinary signature scheme. An aggregate signature is the same length as an ordinary signature in the underlying scheme.

The aggregation algorithm takes as input signatures $\sigma_1,\ldots,\sigma_n$ on respective messages $M_1,\ldots,M_n$ under respective public keys $PK_1,\ldots,PK_n$. (The assignment of indices is arbitrary.) It outputs a single aggregate signature σ.

The aggregate verification algorithm, given an aggregate signature σ, messages $M_1,\ldots,M_n$, and public keys $PK_1,\ldots,PK_n$, verifies that σ is a valid aggregate signature on the given messages under the given keys.

2.1 Bilinear Maps

We start by reviewing the mathematical underpinnings of general aggregate signatures: Gap Diffie-Hellman groups and bilinear groups. Gap Diffie-Hellman groups arise from a separation between Computational and Decision Diffie-Hellman. Bilinear groups arise from the presence of a bilinear map, a function with certain properties.

Consider a multiplicative cyclic group G of prime order p, with generator g. On this group, the familiar Diffie-Hellman problems proceed as follows.

Computational Diffie-Hellman (CDH). Given $g, g^a, h \in G$, compute $h^a \in G$.

Decision Diffie-Hellman (DDH). Given $g, g^a, h, h^b \in G$, decide whether a equals b. Tuples of this form — (g, g^a, h, h^a) — are termed Diffie-Hellman tuples.

Loosely stated, the CDH assumption is that it is computationally infeasible to solve random instances of the CDH problem; the DDH assumption is similarly defined.

GDH Groups. For many choices of group G, such as subgroups of $\mathbb{Z}_q^*$, both the CDH and DDH assumptions are believed to hold. As we will see, however, on certain elliptic-curve groups, the DDH problem is easy to solve, whereas CDH is believed hard [6, 22]. We term groups that have this property Gap Diffie-Hellman (GDH) groups. GDH is an instance of a family of gap problems discussed by Okamoto and Pointcheval [25].

Bilinear groups. Currently, the only known examples of GDH groups have additional structure, namely, a bilinear map. A bilinear map is a map $e : G \times G \rightarrow G_T$ — where G_T is another multiplicative cyclic group of prime order p — with the following properties:

- Computable: there exists an efficiently-computable algorithm for computing $e(u,v)$, for all $u, v \in G$.
- Bilinear: for all $u, v \in G$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u,v)^{ab}$.
- Non-degenerate: $e(g,g) \neq 1$.

A bilinear group is any group that possesses such a map e, and on which CDH is hard.

Joux and Nguyen [15] noted that a bilinear map e provides an algorithm for solving DDH. For a tuple (g, g^a, h, h^b) we have

$$a = b \bmod p \quad \Longleftrightarrow \quad e(h, g^a) = e(h^b, g) \ .$$

Consequently, if a group G is a bilinear group then G is also a GDH group. (The converse is probably not true.)

We now describe the elliptic curve groups mentioned above. Let $E/\mathbb{F}_q$ be an elliptic curve, and let G be a subgroup (of prime order p) of the curve's group of points $E(\mathbb{F}_q)$. On certain curves, the Weil and modified Tate pairings [14, 12, 13] yield a bilinear map $e : G \times G \rightarrow G_T$. The target group G_T is a subgroup of $\mathbb{F}_{q^\alpha}$, where α is a security multiplier that depends on the curve and on the group G.

The multiplier α provides a tradeoff between efficiency and security. The smaller the value of α, the faster is the computation of the bilinear map; the larger the value of α, the more difficult is the CDH problem on G. Current CDH algorithms on G require solving the discrete logarithm problem either in the generic group G (of order p) or in the finite field $\mathbb{F}_{q^\alpha}$ [17, 18]. We note that members of the MNT family of curves [22] have large subgroups with security multiplier $\alpha = 6$, which is suitable for our needs.

2.2 The BLS Signature Scheme

We now describe the BLS short signature scheme. The scheme works in any Gap Diffie-Hellman group G. It requires, in addition, a hash function from the message space onto the group G. The scheme is related to the undeniable signature scheme of Chaum and Pedersen [7].

Specifically, let $G = \langle g \rangle$ be a GDH group of prime order p, with a hash function $H : \{0,1\}^* \to G$, viewed as a random oracle [2]. Any string can be signed; a signature is a single element of G. The scheme comprises the three algorithms below.

Key Generation. Pick random $x \stackrel{R}{\leftarrow} \mathbb{Z}_p$ and compute $v \leftarrow g^x$. The public key is $v \in G$. The private key is $x \in \mathbb{Z}_p$.

Signing. Given a private key x and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M)$, where $h \in G$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G$.

Verification. Given a public key v, a message M, and a signature σ, compute $h \leftarrow H(M)$ and verify that (g, v, h, σ) is a valid Diffie-Hellman tuple.

The intuition is: On a correct signature, $v = g^x$, and $\sigma = h^x$, so (g, v, h, σ) is a Diffie-Hellman tuple. This establishes the validity of the scheme. Its security against existential forgery under a chosen message attack can be shown based on the CDH assumption in G [6].

Signature length. Points on an elliptic curve group $G < E(\mathbb{F}_q)$ are usually represented as a pair (x, y) of elements of $\mathbb{F}_q$, but BLS remains valid and secure even if only the x-coordinate of every signature point $\sigma \in G$ is transmitted. Thus, on an MNT curve (with $\alpha = 6$) over a 170-bit field, BLS signatures are 170 bits long, and provide security comparable to that of 1024-bit RSA [26, 4] or 320-bit DSA [11]. In other words, BLS signatures are half the size of DSA with comparable security.

Because of their simple mathematical structure, BLS signatures are amenable to a variety of extensions, including threshold signatures, multisignatures, and blind signatures [3].

2.3 Bilinear Aggregate Signatures

We now describe the bilinear aggregate signature scheme [5]. Unlike the BLS signature scheme on which it is based, the bilinear aggregate signature scheme requires the group G to be a bilinear group — a general Gap Diffie-Hellman group is insufficient. As in the BLS scheme, any string can be signed. The scheme employs a random oracle hash function, but one that takes both a string and an element of G as input: $H : G \times \{0,1\}^* \to G$.

The bilinear aggregate signature scheme enables general aggregation. An arbitrary aggregating party unrelated to, and untrusted by, the original signers can combine pre-existing signatures into an aggregate. The system does not impose an order on the aggregated elements. Note that, when needed, an order can be imposed by prepending index numbers to the messages being signed.

For notational convenience, we number the users whose signatures are aggregated $1, 2, \ldots, n$ in the description below. This numbering is arbitrary. The number of signatures n in an aggregate is effectively unbounded (viz., polynomial in the security parameter).

The scheme includes the three usual algorithms for generating and verifying individual signatures, as well as two additional algorithms that provide the aggregation capability.

Key Generation. For a particular user, pick random $x \xleftarrow{R} \mathbb{Z}_p$, and compute $v \leftarrow g^x$. The user's public key is $v \in G$. The user's private key is $x \in \mathbb{Z}_p$.

Signing. For a particular user, given the public key v, the private key x, and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(v,M)$, where $h \in G$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G$.

Verification. Given a user's public key v, a message M, and a signature σ, compute $h \leftarrow H(v,M)$; accept if $e(\sigma,g) = e(h,v)$ holds.

Aggregation. Arbitrarily assign to each user whose signature will be aggregated an index i, ranging from 1 to n. Each user i provides a signature $\sigma_i \in G$ on a message $M_i \in \{0,1\}^*$ of her choice. Compute $\sigma \leftarrow \prod_{i=1}^{n} \sigma_i$. The aggregate signature is $\sigma \in G$.

Aggregate Verification. We are given an aggregate signature $\sigma \in G$ for a set of users, indexed as before, and are given the original messages $M_i \in \{0,1\}^*$ and public keys $v_i \in G$. To verify the aggregate signature σ, compute $h_i \leftarrow H(v_i,M_i)$ for $1 \le i \le n$, and accept if $e(\sigma,g) = \prod_{i=1}^{n} e(h_i,v_i)$ holds.

The test employed in the verification of individual signatures is the same DDH test used in BLS verification, but rewritten in bilinear-map notation. Note that a bilinear aggregate signature, like a BLS signature, is a single element of G. Unlike in BLS, the signing process signs both the message and the user's public key.

The intuition behind bilinear aggregate signatures is as follows. User i has a private key $x_i \in \mathbb{Z}_p$ and a public key $v_i = g^{x_i}$. User i's signature, if correctly formed, is $\sigma_i = h_i^{x_i}$, where h_i is the hash of the user's chosen message, M_i, along with her public key v_i. The aggregate signature σ is thus $\sigma = \prod_i \sigma_i = \prod_i h_i^{x_i}$. Using the properties of the bilinear map, the left-hand side of the verification equation expands:

$$\begin{aligned} e(\sigma,g) &= e(\prod_i h_i^{x_i}, g) \\ &= \prod_i e(h_i,g)^{x_i} \\ &= \prod_i e(h_i,g^{x_i}) \\ &= \prod_i e(h_i,v_i) \ , \end{aligned}$$

which is the right-hand side, as required. This establishes the validity of the scheme; its security against forgery can be demonstrated. Even when the would-be forger possesses all but one of the private keys, he cannot frame the remaining honest user. See [5] for the exact security model and proof of security based on CDH in G.

Incremental Aggregation. Consider an aggregate signature σ on messages $M_1,\ldots,M_n$ under public keys $v_1,\ldots,v_n$. An additional signature σ_{n+1} (on a message M_{n+1} under public key v_{n+1}) can be folded into the aggregate: $\sigma' \leftarrow \sigma \cdot \sigma_{n+1}$. If some signature σ_j included in σ is known, it can be removed from the aggregate: $\sigma' \leftarrow \sigma/\sigma_j$. If, however, only the messages, public keys, and the aggregate signature σ are known, recovering the individual signatures $\sigma_1,\ldots,\sigma_n$ from the aggregate is hard. This hardness assumption, the basis for other signature constructions [5], was shown by Coron and Naccache to be equivalent to Computational Diffie-Hellman [10].

3 Sequential Aggregate Signatures

Sequential aggregate signatures are a variant of aggregate signatures. In a sequential aggregate signature scheme, signatures are not individually generated and then combined into an aggregate. Rather, a would-be signer transforms a sequential aggregate into another that includes a signature on a message of his choice. Signing and aggregation are a single operation. Sequential aggregate signatures are built in layers, like an onion; the first signature in the aggregate is the inmost. As with general aggregate signatures, the resulting sequential aggregate is the same length as an

ordinary signature. This behavior closely mirrors the sequential nature of certificate chains in a PKI.

For sequential aggregate signatures, aggregation and signing are performed in a single combined operation. The operation takes as input a private key SK, a message M_i to sign, and a sequential aggregate signature σ' on messages $M_1, \ldots, M_{i-1}$ under respective public keys $PK_1, \ldots, PK_{i-1}$, where M_1 is the inmost message. It adds a signature on M_i under SK to the aggregate, outputting a sequential aggregate σ on all i messages $M_1, \ldots, M_i$.

The aggregate verification algorithm, given a sequential aggregate signature σ, messages $M_1, \ldots, M_i$, and public keys $PK_1, \ldots, PK_i$, verifies that σ is a valid sequential aggregate (with M_1 inmost) on the given messages under the given keys.

3.1 Trapdoor Homomorphic Permutations

Sequential aggregate signatures are built from trapdoor homomorphic permutations. We first review trapdoor permutations and then describe the sequential aggregate scheme to which they give rise.

A permutation family Π is a collection of permutations of some domain D. Each permutation in Π has a description $s \in S$. Anyone given a description s can evaluate the corresponding permutation.

Loosely speaking, a permutation family is one-way if, given a permutation description s, it is infeasible to invert the corresponding permutation. A permutation family is trapdoor if each description s has some corresponding trapdoor $t \in T$ such that it is easy to invert the permutation corresponding to s with t, but infeasible without t. A trapdoor permutation family is necessarily one-way. (Here S and T are arbitrary sets.)

More formally, a trapdoor permutation family Π comprises three algorithms: *Generate*, *Evaluate*, and *Invert*. The randomized generation algorithm *Generate* outputs the description $s \in S$ of a permutation along with the corresponding trapdoor $t \in T$. The evaluation algorithm *Evaluate*, given the permutation description s and a value $x \in D$, outputs $a \in D$, the image of x under the permutation. The inversion algorithm *Invert*, given the permutation description s, the trapdoor t, and a value $a \in D$, outputs the preimage of a under the permutation.

We require that $Evaluate(s,\cdot)$ be a permutation of D for all $(s,t) \xleftarrow{R} Generate$, and that $Invert(s,t,Evaluate(s,x)) = x$ hold for all $(s,t) \xleftarrow{R} Generate$ and for all $x \in D$.

A trapdoor permutation is homomorphic if D is a group with some operation $*$ and if, for all (s,t) generated by *Generate*, the permutation $\pi : D \to D$ induced by $Evaluate(s,\cdot)$ is an automorphism on D. That is, if $a = \pi(x)$ and $b = \pi(y)$, then $a * b = \pi(x * y)$.

When it engenders no ambiguity, we consider the output of the generation algorithm *Generate* as a probability distribution Π on permutations, and write $(\pi, \pi^{-1}) \xleftarrow{R} \Pi$; here π is the permutation $Evaluate(s,\cdot)$, and π^{-1} is the inverse permutation $Invert(s,t,\cdot)$.

It can happen that each permutation $Evaluate(s,\cdot)$ is over a different domain D_s. For example, the RSA permutation family gives permutations over domains $\mathbb{Z}_N^*$, where each user has a distinct modulus N. We consider this further in Section 3.4. For now we assume that all permutations in the family are over the same domain D.

3.2 Full-domain signatures

We review the full-domain hash signature scheme. The scheme, introduced by Bellare and Rogaway [1] and further analyzed by Coron [8], works in any trapdoor permutation family.

Like the others discussed above, the full-domain hash signature scheme employs a random-oracle hash function $H : \{0,1\}^* \to D$. The hash function maps bit strings into the entire domain D (rather than some subset of D), a fact which gives the scheme its name.

Key Generation. For a particular user, pick random $(s,t) \xleftarrow{R} \mathit{Generate}$. The user's public key PK is s. The user's private key SK is (s,t).

Signing. For a particular user, given the private key (s,t) and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M)$, where $h \in D$, and $\sigma \leftarrow \mathit{Invert}(s,t,h)$. The signature is $\sigma \in D$.

Verification. Given a user's public key s, a message M, and a signature σ, compute $h \leftarrow H(M)$; accept if $h = \mathit{Evaluate}(s,\sigma)$ holds.

These algorithms can also be described using the simplified notation given above. A user signs a message by publishing $\sigma = \pi^{-1}(H(M))$; the signature is valid if $\pi(\sigma) = H(M)$ holds.

The signature scheme is secure against existential forgery under a chosen message attack if Π is a trapdoor permutation family [1]. If Π is homomorphic as well, then the security reduction can be made more efficient [8].

3.3 Sequential Aggregate Signatures

We now describe the trapdoor sequential aggregate signature scheme. The scheme is related to the full-domain hash signature scheme, but must be instantiated on a *homomorphic* trapdoor permutation. The scheme is based on a multisignature scheme due to Micali, Ohta, and Reyzin [19].

To simplify the presentation of the scheme, we introduce some notation for vectors. We write a vector as $\mathbf{x}$, its length as $|\mathbf{x}|$, and its elements as $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{|\mathbf{x}|}$. We denote vector concatenation as $\mathbf{x}\|\mathbf{y}$ and appending an element to a vector as $\mathbf{x}\|z$. For a vector $\mathbf{x}$, $\mathbf{x}|_a^b$ is the sub-vector containing elements $\mathbf{x}_a, \mathbf{x}_{a+1}, \ldots, \mathbf{x}_b$. It is necessarily the case that $1 \le a \le b \le |\mathbf{x}|$.

Like the others, this scheme employs a full-domain random-oracle hash function H mapping inputs into D. A signer provides to H every public key and every message in the aggregate signature she is creating. Thus H is of the form $H : \bigcup_{j=1}^{\infty} \left[(S)^j \times (\{0,1\}^*)^j\right] \to D$.

Key Generation. For a particular user, pick random $(s,t) \xleftarrow{R} \mathit{Generate}$. The user's public key PK is s. The user's private key SK is (s,t).

Aggregate Signing. The input is a private key (s,t), a message $M \in \{0,1\}^*$ to be signed, and a sequential aggregate signature σ' on a vector of messages $\mathbf{M}$ under a vector of public keys $\mathbf{s}$. No key may appear twice in $\mathbf{s}$. Furthermore, the vectors $\mathbf{M}$ and $\mathbf{s}$ must have the same length. Let i equal $|\mathbf{M}|$. If i is 0, σ' must equal 1, the unit of D.
Compute $h \leftarrow H(\mathbf{s}\|s,\mathbf{M}\|M)$, where $h \in D$, and $\sigma \leftarrow \mathit{Invert}(s,t,h * \sigma')$. The sequential aggregate signature is $\sigma \in D$.

Aggregate Verification. The input is a sequential aggregate signature σ on messages $\mathbf{M}$ under public keys $\mathbf{s}$, where $|\mathbf{M}| = |\mathbf{s}| = i$. To verify, set $\sigma_i \leftarrow \sigma$. Then, for $j = i, \ldots, 1$, set $\sigma_{j-1} \leftarrow \mathit{Evaluate}(\mathbf{s}_j, \sigma_j) * H(\mathbf{s}|_1^j, \mathbf{M}|_1^j)^{-1}$. Accept if σ_0 equals 1.

Written using π-notation, a sequential aggregate signature is of the form

$$\pi_i^{-1}(h_i * \pi_{i-1}^{-1}(h_{i-1} * \pi_{i-2}^{-1}(\cdots \pi_2^{-1}(h_2 * \pi_1^{-1}(h_1))\cdots))),$$

where $h_j = H(\mathbf{s}|_1^j, \mathbf{M}|_1^j)$. Verification evaluates the permutations in the forward direction, peeling layers away until the center is reached.

The trapdoor sequential aggregate signature scheme is secure against forgery, assuming Π is a homomorphic trapdoor permutation family. Even when the would-be forger possesses all but one of the private keys, he cannot frame the remaining honest user. For the precise security model and proof of security see [27].

3.4 Aggregating with RSA

We consider the details of instantiating the sequential aggregate signature scheme presented above using the RSA permutation family.

The RSA function was introduced by Rivest, Shamir, and Adleman [26]. If $N = pq$ is the product of two large primes and $ed = 1 \bmod \phi(N)$, then $\pi(x) = x^e \bmod N$ is a permutation on $\mathbb{Z}_N^*$, and $\pi^{-1}(x) = x^d \bmod N$ is its inverse. Setting $s = (N, e)$ and $t = (d)$ gives a trapdoor permutation that is multiplicatively homomorphic.

A difficulty arises since two users cannot share the same modulus N. Thus the domains of the one-way permutations belonging to the aggregating users differ, making it difficult to treat RSA as a family of trapdoor permutations. We give two approaches that allow us to create sequential aggregate signatures from RSA nonetheless. The first method imposes more restrictions on the choices of signing keys than the second. Aggregate signatures created by the second method grow by one bit per signature.

Suppose the n users have moduli $N_1, \ldots, N_n$, with N_1 inmost. We assume that the moduli are approximately the same size, i.e., that $\lfloor \log_2 N_1 \rfloor = \lfloor \log_2 N_2 \rfloor = \cdots = \lfloor \log_2 N_n \rfloor$. Let N be the minimum of $N_1, \ldots, N_n$. The hash function H maps into the set $\{1, \ldots, N-1\}$; hashes not in $\mathbb{Z}_{N_i}^*$ for some i can be dealt with by iterating the hash, using the method given by Bellare and Rogaway [1, Section 4].

In the first method, the moduli are constrained so that $N_1 < N_2 < \cdots < N_n$. A sequential aggregate signature σ_i under the keys with moduli $N_1, \ldots, N_i$ is such that $\sigma_i < N_i < N_{i+1}$. Thus (except with negligibly small probability) σ_i is in the domain of the permutation with modulus N_{i+1}. Letting $\pi_i(x) = x^{e_i} \bmod N_i$, we can apply the sequential aggregate signature scheme of Section 3.3 otherwise unchanged.

In the second method, the moduli are not ordered and increasing. It can then happen that σ_i is larger than N_{i+1}. We deal with this by truncating σ_i so that it fits. Let ℓ equal $\lfloor \log_2 N \rfloor$. Then $2^\ell < N_1, \ldots, N_n < 2^{\ell+1}$. Now, if some i-element sequential aggregate signature σ_i is such that $\sigma_i \geq 2^\ell$, we emit the bit $b_i \leftarrow 1$ and continue aggregation using $\sigma_i' \leftarrow \sigma_i - 2^\ell$; otherwise we emit the bit $b_i \leftarrow 0$ and continue aggregation using $\sigma_i' \leftarrow \sigma_i$. The n-bit vector $b_1, \ldots, b_n$ can be appended to the sequential aggregate signature, which then grows by a single bit per aggregating user, or it can be omitted and recovered by an exhaustive search of the 2^n possibilities.

These two schemes are no longer full-domain hash signature schemes, but, since the moduli are all approximately the same size, Coron's partial-domain hash analysis [9] applies to either.

4 Conclusions

We surveyed two techniques for signature aggregation. Both methods provide the ability to compress multiple signatures by distinct signers on distinct messages into a single signature. The first method, based on bilinear maps, provides general aggregation, where anyone can combine signatures into an aggregate at any time, without the cooperation of the signers. The

second method, based on homomorphic trapdoor permutations such as RSA, provides only sequential aggregation where aggregation must be done during the signing process. General aggregation is more a powerful mechanism than sequential aggregation. For example, sequential aggregation can be built from general aggregation. Also, general aggregation seems easier to use.

We discussed two applications for signature aggregation: compressing certificate chains in a PKI and compressing messages in secure routing protocols. Both aggregation techniques are adequate for these applications.

References

[1] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. Denning, R. Pyle, R. Ganesan, R. Sandhu, and V. Ashby, editors, *Proceedings of CCS 1993*, pages 62–73. ACM, 1993.

[2] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In U. Maurer, editor, *Proceedings of Eurocrypt 1996*, volume 1070 of *LNCS*, pages 399–416. Springer-Verlag, 1996.

[3] A. Boldyreva. Efficient threshold signature, multisignature and blind signature schemes based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer-Verlag, 2003.

[4] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–13, 1999.

[5] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In E. Biham, editor, *Proceedings of Eurocrypt 2003*, volume 2656 of *LNCS*, pages 416–32. Springer-Verlag, 2003.

[6] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 514–32. Springer-Verlag, 2001. Full paper: `http://crypto.stanford.edu/~dabo/pubs.html`.

[7] D. Chaum and T. Pedersen. Wallet databases with observers. In E. Brickell, editor, *Proceedings of Crypto 1992*, volume 740 of *LNCS*, pages 89–105. Springer-Verlag, 1992.

[8] J.-S. Coron. On the exact security of full domain hash. In M. Bellare, editor, *Proceedings of Crypto 2000*, volume 1880 of *LNCS*, pages 229–35. Springer-Verlag, 2000.

[9] J.-S. Coron. Security proof for partial-domain hash signature schemes. In M. Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 613–26. Springer-Verlag, 2002.

[10] J.-S. Coron and D. Naccache. Boneh et al.'s *k*-element aggregate extraction assumption is equivalent to the Diffie-Hellman assumption. In C. S. Laih, editor, *Proceedings of Asiacrypt 2003*, LNCS. Springer-Verlag, 2003. To appear.

[11] FIPS 186-2. Digital signature standard, 2000.

[12] G. Frey, M. Muller, and H. Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Trans. Info. Th.*, 45(5):1717–9, 1999.

[13] S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In C. Fieker and D. Kohel, editors, *Proceedings of ANTS V*, volume 2369 of *LNCS*, pages 324–37. Springer-Verlag, 2002.

[14] P. Gaudry, F. Hess, and N. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *J. Cryptology*, 15(1):19–46, 2002.

[15] A. Joux and K. Nguyen. Separating decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. Cryptology ePrint Archive, Report 2001/003, 2001. `http://eprint.iacr.org/`.

[16] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (Secure-BGP). *IEEE J. Selected Areas in Comm.*, 18(4):582–92, April 2000.

[17] U. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In Y. Desmedt, editor, *Proceedings of Crypto 1994*, volume 839 of *LNCS*, pages 271–81. Springer-Verlag, 1994.

[18] A. Menezes, T. Okamoto, and P. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Info. Th.*, 39(5):1639–46, 1993.

[19] S. Micali, K. Ohta, and L. Reyzin. Provable-subgroup signatures. Unpublished manuscript, 1999.

[20] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures (extended abstract). In *Proceedings of CCS 2001*, pages 245–54. ACM Press, 2001.

[21] S. Micali and R. Rivest. Transitive signature schemes. In *Proceedings of RSA 2002*, volume 2271 of *LNCS*, pages 236–43. Springer-Verlag, 2002.

[22] A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Trans. Fundamentals*, E84-A(5):1234–43, May 2001.

[23] K. Ohta and T. Okamoto. Multisignature schemes secure against active insider attacks. *IEICE Trans. Fundamentals*, E82-A(1):21–31, 1999.

[24] T. Okamoto. A digital multisignature scheme using bijective public-key cryptosystems. *ACM Trans. Computer Systems*, 6(4):432–41, November 1988.

[25] T. Okamoto and D. Pointcheval. The gap problems: A new class of problems for the security of cryptographic primitives. In K. Kim, editor, *Proceedings of PKC 2001*, volume 1992 of *LNCS*, pages 104–18. Springer-Verlag, 2001.

[26] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Commun. ACM*, 21:120–126, 1978.

[27] H. Shacham. Sequential aggregate signatures from trapdoor homomorphic permutations. Cryptology ePrint Archive, Report 2003/091, 2003. `http://eprint.iacr.org/`.

II. On the Cost of Factoring RSA-1024

Adi Shamir Eran Tromer

Weizmann Institute of Science
{shamir,tromer}@wisdom.weizmann.ac.il

Abstract

As many cryptographic schemes rely on the hardness of integer factorization, exploration of the concrete costs of factoring large integers is of considerable interest. Most research has focused on PC-based implementations of factoring algorithms; these have successfully factored 530-bit integers, but practically cannot scale much further. Recent works have placed the bottleneck at the sieving step of the Number Field Sieve algorithm. We present a new implementation of this step, based on a custom-built hardware device that achieves a very high level of parallelism "for free". The design combines algorithmic and technological aspects: by devising algorithms that take advantage of certain tradeoffs in chip manufacturing technology, efficiency is increased by many orders of magnitude compared to previous proposals. Using this hypothetical device (and ignoring the initial R&D costs), it appears possible to break a 1024-bit RSA key in one year using a device whose cost is about $10M (previous predictions were in the trillions of dollars).

1 Introduction

The security of many cryptographic schemes and protocols depends on the hardness of finding the factors of large integers drawn from an appropriate distribution. The best known algorithm for factoring large integers is the Number Field Sieve (NFS)[1], whose time and space complexities are subexponential in the size of the composite. However, little is known about the real complexity of this problem. The evident confidence in the hardness of factoring comes from observing that despite enormous interest, no efficient factoring algorithm has been found.

To determine what key sizes are appropriate for a given application, one needs concrete estimates for the cost of factoring integers of various sizes. Predicting these costs has proved notoriously difficult, for two reasons. First, the performance of modern factoring algorithms is not understood very well: their complexity analysis is often asymptotic and heuristic, and leaves large uncertainty factors. Second, even when the exact algorithmic complexity is known, it is hard to estimate the concrete cost of a suitable hypothetical large-scale computational effort using current technology; it's even harder to predict what this cost would be at the end of the key's planned lifetime, perhaps a decade or two into the future.

Due to these difficulties, common practice is to rely on extrapolations from past factorization experiments. Many such experiments have been performed and published; for example, the successful factorization of a 512-bit RSA key in 1999 [5] clearly indicated

[1] See [10] for the seminal works and [17] for an introduction. The subtask we discuss is defined in Section 2.1.

the insecurity of such keys for many applications, and prompted a transition to 1024-bit keys (often necessitating software or hardware upgrades).[2] The current factorization record, obtained nearly four years later in March 2003, stands at 530 bits [1].[3] From this data, and in light of the subexponential complexity of the algorithm used, it seems reasonable to surmise that factoring 1024-bit RSA keys, which are currently in common use, should remain infeasible for well over a decade.

However, the above does not reflect a fundamental economy-of-scale consideration. While the published experiments have employed hundreds of workstations and Cray supercomputers, they have always used general-purpose computer hardware. However, when the workload is sufficiently high (either because the composites are large or because there are many of them to factor), it becomes more efficient to construct and employ custom-built hardware dedicated to the task. Direct hardware implementation of algorithms is considerably more efficient than software implementations, and makes it possible to eliminate the expensive yet irrelevant peripheral hardware found in general-purpose computers. An example of this approach is the EFF DES Cracker [7], built in 1998 at a cost of $210,000 and capable of breaking a DES key in expected time of 4.5 days using 36864 search units packed into 1536 custom-built gate array chips. Indeed, its equipment cost per unit of throughput was much lower than similar experiments that used general-purpose computers.

Custom-built hardware can go beyond efficient implementation of standard algorithms — it allow specialized data paths, enormous parallelism and can even use non-electronic physical phenomena. Taking advantage of these requires new algorithms or adaptation of existing ones. One example is the TWINKLE device [18, 13], which implements the sieving step of the NFS factoring algorithm using a combination of highly parallel electronics and an analog optical adder.

Recently, D. J. Bernstein made an important observation [3] about the major algorithmic steps in the NFS algorithm. These steps have a huge input, which is accessed over and over many times. Thus, traditional PC-based implementations are very inefficient in their use of storage: a huge number of storage bits is just sitting in memory, waiting for a single processor to access them. Most of the previous work on NFS cost analysis (with the notable exception of [21]) considered only the number of processor instructions, which is misleading because the cost of memory greatly outweighs the cost of the processor. Instead, one should consider the equipment cost per unit of throughput, i.e., the construction cost multiplied by the running time per unit of work.

Following this observation, Bernstein presented a new parallel algorithm for the matrix step of the NFS algorithm, based on a mesh-connected array of processors. Intuitively, the idea is to attach a simple processor to each block of memory and execute a distributed algorithm among these processors to get better utilization of the memory. With this algorithm, and by changing some adjustable parameter in the NFS algorithm so as to minimize "cost per unit of throughput" rather than instruction count, Bernstein's algorithm allows one to factor integers that are 3.01 times longer compared to traditional algorithms (though only 1.17 times longer when the traditional algorithms are re-optimized for throughput cost). Subsequent works [14, 8] evaluated the practicality of Bernstein's algorithm for 1024-bit composites, and suggested improved versions that significantly reduced its cost. With these hypothetical (but detailed) designs, the cost of the matrix step was brought down from trillions of dollars [21] to at most a few dozen million dollars (all figures are for completing the task in 1 year).

This left open the issue of the other major step in the Number Field Sieve, namely the sieving step.

[2] Earlier extrapolations indeed warned of this prospect.

[3] Better results were obtained for composites of a special form, using algorithms which are not applicable to RSA keys.

For 1024-bit composites it was predicted that sieving would require trillions of dollars,[21][4] and would be impractical even when using the TWINKLE device. This article discusses a new design for a custom-hardware implementation of the sieving step, which reduces this cost to about $10M. The new device, called TWIRL[5], can be seen as an extension of the TWINKLE device. However, unlike TWINKLE it does not have optoelectronic components, and can thus be manufactured using standard VLSI technology on silicon wafers. The underlying idea is to use a single copy of the input to solve many subproblems in parallel. Since input storage dominates cost, if the parallelization overhead is kept low then the resulting speedup is obtained essentially for free. Indeed, the main challenge lies in achieving this parallelism efficiently while allowing compact storage of the input. Addressing this involves myriad considerations, ranging from number theory to VLSI technology. The resulting design is sketched in the following sections, and a more detailed description appears in [19].

2 Context

2.1 The Sieving Task

The TWIRL device is specialized to a particular task, namely the sieving task which occurs in the Number Field Sieve (and also in its predecessor, the Quadratic Sieve). This section briefly reviews the sieving problem, with many simplifications.

The inputs of the sieving problem are $R \in \mathbb{Z}$ (*sieve line width*), $T > 0$ (*threshold*) and a set of pairs (p_i,r_i) where the p_i are the prime numbers smaller than some *factor base bound* B. There is, on average, one pair per such prime. Each pair (p_i,r_i) corresponds to an arithmetic progression $P_i = \{a : a \equiv r_i \pmod{p_i}\}$. We are interested in identifying the sieve locations $a \in \{0,\ldots,R-1\}$ that are members of many progressions P_i with large p_i:

$$g(a) > T \quad \text{where} \quad g(x) = \sum_{i:a\in P_i} \log_h p_i$$

for some small constant h. It is permissible to have "small" errors in this threshold check; in particular, we round all logarithms to the nearest integer. For each a that exceeds the threshold, we also need to find the set $\{i : a \in P_i\}$ of progressions that contribute to $g(a)$.

We shall concentrate on 1024-bit composites and a particular choice of the adjustable NFS parameters, with $R = 1.1 \cdot 10^{15}$ and $B = 3.5 \cdot 10^9$. We need to perform $H = 2.7 \cdot 10^8$ such sieving tasks, called *sieve lines*, that have different (though related) inputs.[6] The numerical values that appear below refer to this specific parameter choice.

2.2 Traditional Sieving

The traditional method of performing the sieving task is a variant of Eratosthenes's algorithm for finding primes. It proceeds as follows. An array of accumulators $C[a]$ is initialized to 0. Then, the progressions P_i are considered one by one, and for each P_i the indices $a \in P_i$ are calculated and the value $\log_h p_i$ is added to every such $C[a]$. Finally, the array is scanned to find the a values where $C[a] > T$. The point is that when looking at a specific P_i its members can be enumerated very efficiently, so the amortized cost of a $\log_h p_i$ contribution is low.

When this algorithm is implemented on a PC, we cannot apply it to the full range $a = 0,\ldots,R-1$ since

[4] [15] gave a lower bound of about $160M for a one-day effort. This disregarded memory, but is much closer to our results since the new device greatly reduces the amortized cost of memory.

[5] TWIRL stands for The Weizmann Institute Relation Locator.

[6] In fact, for each sieve line we need to perform two sieves: a "rational sieve" and an "algebraic sieve" (see Section 3.3). The parameters given here correspond to the rational sieve, which is responsible for most (two thirds) of the device's cost.

there would not be enough RAM to store R accumulators. Thus, the range is broken into smaller chunks, each of which is processed as above. However, if the chunk size is not much larger than B then most progressions make very few contributions (if any) to each chunk, so the amortized cost per contribution increases. Thus, a large amount of memory is required, both for the accumulators and for storing the input (that is, the list of progressions). As Bernstein [3] observed, this is inherently inefficient because each memory bit is accessed very infrequently.

2.3 Sieving with TWINKLE

An alternative way of performing the sieving was proposed in the TWINKLE device [18, 13], which operates as follows. Each TWINKLE device consists of a wafer containing numerous independent cells, each in charge of a single progression P_i. After initialization the device operates synchronously for R clock cycles, corresponding to the sieving range $\{0 \le a < R\}$. At clock cycle a, the cell in charge of the progression P_i emits the value $\log_h p_i$ iff $a \in P_i$. The values emitted at each clock cycle are summed to obtain $g(x)$, and if this sum exceeds the threshold T then the integer a is reported. This event is announced back to the cells, so that the i values of the pertaining P_i is also reported.

The global summation is done using analog optics: to "emit" the value $\log p_i$, a cell flashes an internal LED whose intensity is proportional to $\log p_i$. A light sensor above the wafer measures the total light intensity in each clock cycle, and reports a success when this exceeds a given threshold. The cells themselves are implemented by simple registers and ripple adders. To support the optoelectronic operations, TWINKLE uses Gallium Arsenide wafers (alas, these are relatively small, expensive and hard to manufacture compared to silicon wafers, which are readily available). Compared to traditional sieving, TWINKLE exchanges the roles of space and time:

	Traditional	**TWINKLE**
Sieve locations	Space (accumulators)	Time
Progressions	Time	Space (cells)

3 TWIRL

3.1 Approach

The TWIRL device follows the time-space reversal of TWINKLE, but increases the throughput by simultaneously processing thousands of sieve locations at each clock cycle. Since this is done with (almost) no duplication of the input, the equipment cost per unit of throughput decreases dramatically. Equivalently, we can say that the cost of storing the huge input is amortized across many parallel processes.

As a first step toward TWIRL, consider an electronic variant of TWINKLE which still operates at a rate of one sieve location per clock cycle, but does so using a pipelined systolic chain of electronic adders.[7] Such a device would consist of a long unidirectional bus, 10 bits wide, that connects millions of conditional adders in series. Each conditional adder is in charge of one progression P_i; when activated by an associated timer, it adds the value $\log_h p_i$ to the bus. At time t, the z-th adder handles sieve location $t - z$. The first value to appear at the end of the pipeline is $g(0)$, followed by $g(1), \ldots, g(R)$, one per clock cycle. See Fig. 1(a).

The parallelization is obtained by handling the sieve range $\{0, \ldots, R-1\}$ in consecutive chunks of length $s = 4096$.[8] To do so, the bus is thickened by a factor of s and now contains s logical lines, where each line carries 10-bit numbers. At time t, the z-th stage of the pipeline handles the sieve locations $(t-z)s+i$,

[7] This variant was considered in [13], but deemed inferior in that context.

[8] $s = 4096$ applies to the rational sieve. For the algebraic sieve (see Section 3.3) we use even higher parallelism: $s = 32768$.

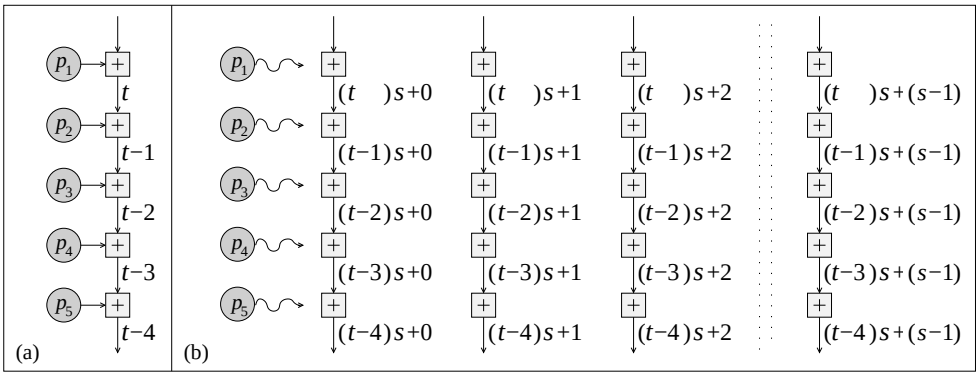

Figure 1: Flow of sieve locations through the device in (a) a chain of adders and (b) TWIRL.

$i \in \{0,\ldots,s-1\}$. The first values to appear at the end of the pipeline are $\{g(0),\ldots,g(s-1)\}$; they appear simultaneously, followed by successive disjoint groups of size s, one group per clock cycle. See Fig. 1(b).

We now have to add the $\log_h p_i$ contributions to all s lines in parallel. Obviously, the naive solution of duplicating all the adders s times gains nothing in terms of equipment cost per unit of throughput. If we try to use the TWINKLE-like circuitry without duplication, we encounter difficulties in scheduling and communicating the contributions across the thick bus: the sieve locations flow down the bus (in Fig. 1(b), vertically), and the contributions should somehow travel across the bus (horizontally) and reach an appropriate adder at exactly the right time.

Accordingly, we replace the simple TWINKLE-like cells by other units that perform scheduling and routing. Each such unit, called a *station*, handles some small portion of the progressions; its interface consists of bus input, bus output, clock and some circuitry for loading the inputs. The stations are connected serially in a pipeline, and at the end of the bus (i.e., at the output of the last station) we place a threshold check unit that produces the device output.

While the function of all the stations is identical, we use a heterogeneous architecture that employs three different station designs — the p_i come in a very large range of sizes, and different sizes involve very different design tradeoffs. The progressions are partitioned into stations according to the size of their intervals p_i, and the optimal station design is employed in each case.

Due to space limitations, we describe only the most important station design, which is used for the majority of progressions. The other station designs, and additional details, can be found in [19].

3.2 Large primes

For every prime smaller than $B = 3.5 \cdot 10^9$ there is (on average) one progression. Thus the majority of progressions have intervals p_i that are much larger than $s = 4096$, so they produce $\log_h p_i$ contributions very seldom. For 1024-bit composites there is a huge number (about $1.6 \cdot 10^8$) of such progressions; even with TWINKLE's simple emitter cells, we could not fit all of them into a single wafer. The primary considera-

tion is thus to store these progressions as compactly as possible, while maintaining a low cost per contribution. Indeed, we will succeed in storing these progressions in compact DRAM-type memory using only sequential (and thus very efficient) read/write access. This necessitates additional support logic, but its cost is amortized across many progressions. This efficient storage lets us fit 4 independent 1024-bit TWIRL devices (each of which is $s = 4096$ times faster than TWINKLE) into a single 30cm silicon wafer.

The station design for these progressions (namely, those with $p_i > 5.2 \cdot 10^5$) is shown in Fig. 2 (after some simplifications). The progressions are partitioned into $8{,}490$ memory banks, so that each bank contains many (between 32 and $2.2 \cdot 10^5$) progressions. Each progression is stored in one of these memory banks, where at any given time it is represented by an *event* of the form (p_i, ℓ_i, τ_i), whose meaning is: "`at time` τ_i`, send a` $\log_h p_i$ `contribution to bus line` ℓ_i`.`"

Each memory bank is connected to a special-purpose processor, which continuously processes these events and sends corresponding *emissions* of the form "`add` $\log_h p_i$ `to bus line` ℓ_i" to attached delivery lines, which span the bus. Each delivery line acts as a shift register that carries the emissions across the bus. Additionally, at every intersection between a delivery line and a bus line there is a conditional adder[9]; when the emission reaches its destination bus line ℓ_i, the value $\log_h p_i$ is added to the value that passes through that point of the bus pipeline at that moment.

Thus, sieve locations are (logically) flowing down the bus at a constant velocity, and emissions are being sent across the bus at a constant velocity. To ensure that each emission "hits" its target at the right time, the two perpendicular flows must be perfectly synchronized, which requires a lot of care. However, the benefit is that the cost per contribution is very low: most of the time the event is stored very compactly in the form of an event in DRAM; then, for a brief moment it occupies the processor, and finally it occupies a delivery line for the minimum possible duration — the amount of time needed to travel across the bus to the destination bus line.

It is the processor's job to ensure accurate scheduling of emissions.[10] The ideal way to achieve this would be to store the events in a priority queue that is sorted by the emission time τ_i. Then, the processor would simply repeat the following loop:[11]

1. Pop the next event (p_i, ℓ_i, τ_i) from the priority queue.
2. Wait until time τ_i and then send an emission to the delivery line, addressed to bus line ℓ_i.
3. Compute the next event (p_i, ℓ_i', τ_i') of this progression, and push it into the priority queue.

Standard implementations of priority queues (e.g., the heap data structure) are unsuitable for our purposes, due to the passive nature of standard DRAM and high latency. First, the processor would need to make a logarithmic number of memory accesses at each iteration. Worse yet, these memory accesses occur at unpredictable places, and thus incur a significant random-access overhead. Fortunately, by taking advantage of the unique properties of the sieving problem we can get a good approximation of a priority queue that is highly efficient.

Briefly, the idea is as follows. The events are read sequentially from memory (step 1 above) in a cyclic order, at constant rate. When the new calculated event

[9] We use carry-save adders, which are very compact and have low latency (the tradeoff is that the bus lines now use a redundant representation of the sums, which doubles the bit-width of the bus).

[10] In the full design [19], there is an additional component, called a *buffer*, which performs fine-tuning and load balancing.

[11] For simplicity, here we ignore the possibility of collisions.

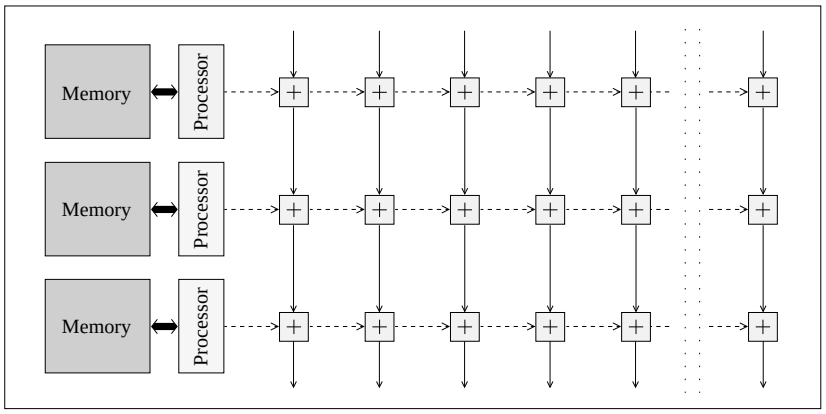

Figure 2: Schematic structure of a (simplified) largish station.

is written back to memory (step 3 above), it is written to a memory address that will be read just before its schedule time τ_i'. Since both τ_i' and the read schedule are known, this memory address is easily calculated by the processor. In this way, after a short stabilization period the processor always reads imminent events,[12] exactly as desired. Each iteration now involves just one sequential-access read operation and one random-access write operation. In addition, it turns out that with appropriate choice of parameters we can cause the write operations to always occur in a small window of activity, just behind the "read head". We may thus view the $8{,}490$ memory banks as closed rings of various sizes, with an active window "twirling" around each ring at a constant linear velocity. Each such sliding window is handled by a fast SRAM-based cache, whose content is swapped in and out of DRAM in large blocks. This allows the bulk of events to be held in DRAM. Better yet, now the only interface to the DRAM memory is through the SRAM cache; this allows elimination of various peripheral circuits that are needed in standard DRAM.

[12] Collisions are handled by adding appropriate slacks.

3.3 Other Highlights

Other station designs. For progressions with small interval ($p_i < 5.2 \cdot 10^5$), it is inefficient to continuously shuttle the progression state to and from passive memory. Thus, each progression is handled by an independent active *emitter* cell that includes an internal counter (similarly to TWINKLE). An emitter serves multiple bus lines, using a variant of the delivery lines described above. Using certain algebraic tricks, these cells can be made very compact. Two such station designs are used: for the progressions with medium-sized intervals, many progressions share the same delivery lines (since emissions are still not very frequent); this requires some coordination logic. For very small intervals, each emitter cell has its own delivery line.

Diaries. Recall that in addition to finding the sieve locations a whose contributions exceed the threshold, we also want to find the sets $\{i : a \in P_i\}$ of relevant progressions. This is accomplished by adding a *diary* to each processor (it suffices to handle the progressions with large interval). The diary is a memory bank which records every emission sent by the processor and saves it for a few thousand clock cycles — the depth of the bus pipeline. By that time, the

corresponding sieve location a has reached the end of the bus and the accumulated sum of logarithms $g(a)$ was checked. If the threshold was exceeded, this is reported to all processors and the corresponding diary entries are recalled and collected. Otherwise, these diary entries are discarded (i.e., their memory is reused).

Cascading the sieves. In the Number Field Sieve we have to perform two sieving tasks in parallel: a *rational sieve* whose parameters were given above, and an *algebraic sieve* which is usually more expensive since it has a large value of B. However, we succeed in greatly reducing the cost of the algebraic sieve by using an even higher parallelization factor for it: $s = 32{,}768$. This is made possible by an alteration that greatly reduces the bus width: the algebraic sieve needs only to consider the sieve locations that passed the rational sieve, i.e., about one in 5,000. Thus we connect the input of the algebraic sieve to the output of the rational sieve, and in the algebraic sieve we replace the thick bus and delivery lines by units that consider only the sieve locations that passed the rational sieve. We now have a much narrower bus containing only 32 lines, though each line now carries both a partial sum (as before) and the index a of the sieve location to which the sum belongs. Logically, the sieve locations still travel in chunks of size s, so that the regular and predictable timing is preserved. Physically, only the "relevant" locations (at most 32) in each chunk are present; emissions addressed to the rest are discarded.

Fault tolerance. The issue of fault tolerance is very important, as silicon wafers normally have multiple local faults. When the wafer contains many independent small chips, one usually discards the faulty ones. However, for 1024-bit composites TWIRL is a wafer-scale design and thus must operate in the presence of faults. All large components of TWIRL can be made fault-tolerant by a combination of techniques: routing around faults, post-processing and re-assigning faulty units to spare. We can tolerate occasional transient faults since the sieving task allows a few errors; only the total number of good a values matters.

4 Cost

Based on the detailed design, we estimated the cost and performance of the TWIRL device using today's VLSI technology (namely, the 0.13μm process used in many modern memory chips and CPUs). While these estimates are hypothetical, they rely on a detailed analysis and should reasonably reflect the real cost. It should be stressed that the NFS parameters assumed are partially based on heuristic estimates. See [19] for details.

1024-bit composites. Recall that to implement NFS we have to perform two different sieving tasks, a rational sieve and an algebraic sieve, which have different parameters. Here, the rational sieve (whose parameters were given above) dominates the cost. For this sieve, a TWIRL device requires $15{,}960\mathrm{mm}^2$ of silicon wafer area, so we can fit 4 of them on a 30cm silicon wafer. Most of the device area is occupied by the large progressions (and specifically, 37% of the device is used for their DRAM banks). For the algebraic sieves we use a higher parallelization factor, $s = 32{,}768$. One algebraic TWIRL device requires $65{,}900\mathrm{mm}^2$ of silicon wafer area — a full wafer — and here too most of the device is occupied by the largish progressions (the DRAM banks occupy 66%).

The devices are assembled in clusters that consist of 8 rational TWIRLs (occupying two wafers) and 1 algebraic TWIRL (on a third wafer), where each rational TWIRL has a unidirectional link to the algebraic TWIRL over which it transmits 12 bits per clock cycle. A cluster handles a full sieve line in $R/32{,}768$ clock cycles, i.e., 33.4 seconds when clocked at 1GHz. The full sieving involves H sieve lines, which would require 194 years when using a single cluster (after a heuristic that rules out 33% of the sieve locations). At a cost of $\$2.9$M (assuming $5,000 per wafer), we can build 194 independent TWIRL clusters that, when run in parallel, would complete the sieving task within 1 year.

After accounting for the cost of packaging, power supply and cooling systems, adding the cost of PCs for collecting the data and leaving a generous error margin,[13] it appears realistic that all the sieving required for factoring 1024-bit integers can be completed within 1 year by a device that costs $\$10\text{M}$ to manufacture. In addition to this per-device cost, there would be an initial NRE cost on the order of $\$20\text{M}$ (for design, simulation, mask creation, etc.).

512-bit composites. Since 512-bit factorization is well-studied [18, 13, 8] and backed by experimental data [5], it is interesting to compare 512-bit TWIRL to previous designs. We shall use the same 512-bit parameters as in [13, 8], though they are far from optimal for TWIRL. With $s = 1{,}024$, we can fit 79 TWIRLs into a single silicon wafer; together, they would handle a sieve line in 0.00022 seconds (compared to 1.8 seconds for TWINKLE wafer and 0.36 seconds for a full wafer using mesh-based design of [8]). Thus, in factoring 512-bit composites the basic TWIRL design is about $1{,}600$ times more cost effective than the best previously published design [8], and $8{,}100$ times more cost effective than TWINKLE. Such a wafer full of TWIRLs, which can be manufactured for about $5,000 in large quantities, can complete the sieving for 512-bit composites in under 10 minutes (this is before TWIRL-specific optimizations which would halve the cost, and using the standard but suboptimal parameter choice).

768-bit composites. For 768-bit composites, a single wafer containing 6 TWIRL clusters can complete the sieving in 95 days. This wafer would cost about $5,000 to manufacture — one tenth of the RSA-768 challenge prize [20]. Unfortunately these figures are not easy to verify experimentally, nor do they provide a quick way to gain $45,000, since the initial NRE cost remains $10M-$20M.

[13] It is a common rule of thumb to estimate the total cost as twice the silicon cost; to be conservative, we triple it.

5 Conclusions

It has been often claimed that 1024-bit RSA keys are safe for the next 15 to 20 years, since when applying the Number Field Sieve to such composites both the sieving step and the linear algebra step would be unfeasible (e.g., [4, 21] and a NIST guideline draft [16]). However, these estimates relied on PC-based implementations. We presented a new design for a custom-built hardware implementation of the sieving step, which relies on algorithms that are highly tuned for the available technology. With appropriate settings of the NFS parameters, this design reduces the cost of sieving to about $10M (plus a one-time cost of $20M). Recent works [14, 9] indicate that for these NFS parameters, the cost of the matrix step is even lower.

Our estimates are hypothetical and rely on numerous approximations; the only way to learn the precise costs involved would be to perform a factorization experiment. However, it is difficult to identify any specific issue that may prevent a sufficiently motivated and well-funded organization from applying the Number Field Sieve to 1024-bit composites within the next few years. This should be taken into account by anyone planning to use a 1024-bit RSA key.

Acknowledgment. This work was inspired by Daniel J. Bernstein's insightful work on the NFS matrix step, and its adaptation to sieving by Willi Geiselmann and Rainer Steinwandt. We thank the latter for interesting discussions of their design and for suggesting an improvement to ours. We are indebted to Arjen K. Lenstra for many insightful discussions, and to Robert D. Silverman, Andrew "bunnie" Huang, Michael Szydlo and Markus Jakobsson for valuable comments and suggestions. Early versions of [12] and the polynomial selection programs of Jens Franke and Thorsten Kleinjung were indispensable in obtaining refined estimates for the NFS parameters.

References

[1] F. Bahr, J. Franke, T. Kleinjung, M. Lochter, M. Böhm, *RSA-160*, e-mail announcement, Apr. 2003, http://www.loria.fr/~zimmerma/records/rsa160

[2] Daniel J. Bernstein, *How to find small factors of integers*, manuscript, 2000, http://cr.yp.to/papers.html

[3] Daniel J. Bernstein, *Circuits for integer factorization: a proposal*, manuscript, 2001, http://cr.yp.to/papers.html

[4] Richard P. Brent, *Recent progress and prospects for integer factorisation algorithms*, proc. COCOON 2000, LNCS **1858** 3–22, Springer-Verlag, 2000

[5] S. Cavallar, B. Dodson, A.K. Lenstra, W. Lioen, P.L. Montgomery, B. Murphy, H.J.J. te Riele, et al., *Factorization of a 512-bit RSA modulus*, proc. Eurocrypt 2000, LNCS **1807** 1–17, Springer-Verlag, 2000

[6] Don Coppersmith, *Modifications to the number field sieve*, Journal of Cryptology, **6**(3) 169–180, 1993

[7] Electronic Frontier Foundation, *DES Cracker Project*, http://www.eff.org/descracker

[8] Willi Geiselmann, Rainer Steinwandt, *A dedicated sieving hardware*, proc. PKC 2003, LNCS **2567** 254–266, Springer-Verlag, 2002

[9] Willi Geiselmann, Rainer Steinwandt, *Hardware to solve sparse systems of linear equations over GF(2)*, proc. CHES 2003, LNCS, Springer-Verlag, to appear.

[10] Arjen K. Lenstra, H.W. Lenstra, Jr., (eds.), *The development of the number field sieve*, Lecture Notes in Math. **1554**, Springer-Verlag, 1993

[11] Arjen K. Lenstra, Bruce Dodson, *NFS with four large primes: an explosive experiment*, proc. Crypto '95, LNCS **963** 372–385, Springer-Verlag, 1995

[12] Arjen K. Lenstra, Bruce Dodson, James Hughes, Wil Kortsmit, Paul Leyland, *Factoring estimates for a 1024-bit RSA modulus*, proc. Asiacrypt 2003, LNCS, Springer-Verlag, to appear.

[13] Arjen K. Lenstra, Adi Shamir, *Analysis and optimization of the TWINKLE factoring device*, proc. Eurocrypt 2002, LNCS **1807** 35–52, Springer-Verlag, 2000

[14] Arjen K. Lenstra, Adi Shamir, Jim Tomlinson, Eran Tromer, *Analysis of Bernstein's factorization circuit*, proc. Asiacrypt 2002, LNCS **2501** 1–26, Springer-Verlag, 2002

[15] Arjen K. Lenstra, Eric R. Verheul, *Selecting cryptographic key sizes*, Journal of Cryptology, **14**(4) 255–293, 2002

[16] NIST, *Key management guidelines, Part 1: General guidance (draft)*, Jan. 2003, http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html

[17] Carl Pomerance, *A Tale of Two Sieves*, Notices of the AMS, 1473–1485, Dec. 1996

[18] Adi Shamir, *Factoring large numbers with the TWINKLE device (extended abstract)*, proc. CHES'99, LNCS **1717** 2–12, Springer-Verlag, 1999

[19] Adi Shamir, Eran Tromer, *Factoring large numbers with the TWIRL device*, proc. Crypto 2003, LNCS **2729**, Springer-Verlag, 2003

[20] RSA Security, *The new RSA factoring challenge*, web page, Jan. 2003, http://www.rsasecurity.com/rsalabs/challenges/factoring/

[21] Robert D. Silverman, *A cost-based security analysis of symmetric and asymmetric key lengths*, Bulletin 13, RSA Security, 2000, http://www.rsasecurity.com/rsalabs/bulletins/bulletin13.html

III. Physical One-Way Functions

Ravikanth S. Pappu *

Abstract

How can we assign unique, tamper-resistant, and unforgeable identifiers to everyday objects at a very low cost? Physical One-Way Functions (POWFs) provide a novel approach to answering this question. POWFs can be obtained from the inherent three-dimensional microstructure of a large class of physical systems known as mesoscopic systems. They are inexpensive to fabricate and prohibitively difficult to duplicate; they admit no compact mathematical representation and are intrinsically tamper-resistant. In this paper, we show how POWFs are obtained by using coherent scattering of visible laser radiation from inhomogeneous structures and experimentally demonstrate their properties. We also discuss potential attacks on POWFs and possible applications.

1 Introduction

Humans have long used physical structures to authenticate objects of value. As early as the 4th millennium BC, the Mesopotamian civilization was using cylindrical seals to certify the contents of envelopes, waybills, ceramics, and bricks. These seals, small cylindrical stones carved with a decorative pattern, were rolled over wet clay to mark the target object. Their use was contemporaneous with the use of clay tablets in everyday life and lasted over two thousand years [4].

Modern banknotes incorporate a variety of different structural features that aid the goals of authentication and anti-counterfeiting. Among these are security threads, hologram foils, iridescent stripes, color-shifting inks, and as proposed recently, radio-frequency identification tags [25]. The number and complexity of security features included on banknotes is an indication of the increasing capability of forgers to reproduce highly specialized features with very low cost equipment. While forgers of yesteryear needed access to expensive printing equipment and skilled engravers, highly sophisticated two-dimensional reprographic systems are easily available to the general public today.

Fundamentally, two major changes have occurred since ancient Mesopotamia. First, the creation of complicated two-dimensional structures with specific properties no longer requires the skill that it once did. Second, the manufacturing and digital revolutions have allowed forgers to stop worrying about the structural features and focus on the logical content (e.g., the denomination of currency as opposed to the physical banknote) of the forged object. Because it is much easier to work with bits than it is with atoms, the asymmetry in effort between the ”good guys” and ”bad guys” and the time lag between the original object and a high-quality forgery has decreased substantially.

The search for uncloneable and tamper-evident physical structures leads ultimately to the theoretically compelling concept of Quantum Money [1]. The key idea here is to augment banknotes with a number of isolated two-state quantum systems, such as spin 1/2 nuclei or photons with orthogonal polarizations,

*Research carried out at the MIT Media Laboratory; author may be reached at ravi@thingmagic.com

which are encoded with the identity of the note. In order to successfully forge the note, a forger has to prepare a counterfeit banknote in the same quantum state as the original. This is theoretically impossible [24]. There are two principal attributes of Quantum Money that make it substantially different from all previous methods of physical authentication. First, the security is provable via the quantum no-cloning theorem which states that an arbitrary, unknown quantum state cannot be cloned with certainty. Second, it makes an explicit connection between physical authentication and the framework of modern cryptography. Practically speaking, however, quantum decoherence, i.e., the loss of the quantum identity of isolated quantum monetary systems by interacting with the environment, prevents any useful realization of the concept. While Quantum Money is not a POWF, it does provide a clear example of a physical authentication system whose non-clonability is provable.

This is the context in which we situate POWFs.

2 Motivation: arms control treaties

In this section, we provide an example where POWFs may be used with benefit.

Arms control treaties typically place numerical limits on treaty-limited weapons systems. As opposed to treaties which ban certain types of weapons outright, treaty-limited items (TLIs) require a tagging system to ensure that more than the allowed number of items exist at any given time [9, 10]. Treaty verification then consists of verifying that the total number of items is below the limit established for that item under the treaty.

The goals of the tagging system are unique: it must provide unambiguous verification of TLIs without allowing the monitoring party undue advantage in tracking the weapons systems for purposes of intelligence gathering and espionage. The requirements on the tag system are discussed at length in [10] and are reproduced here: (a) it must be impossible to copy the tag without detection (b) it must be impossible to spoof the tagging system or to fool it into thinking that a valid tag exists where there actually is none (c) it must not be possible to move the tag from one weapon to another without the knowledge of the monitoring party (d) the tagging system must not aid the monitoring party in locating the weapons in real time (e) the tag should only reveal information required for purposes of verification (f) the system must be reliable and have a low false alarm rate (g) the physical size and the power requirements of the tag must be minimal (h) the tag must be reliable in the range of environments that the weapon is exposed to (i) the system should be inexpensive.

Further, the process of creating and reading tags cannot contain any secrets and a complete description of the tag reading process must be written into the language of the treaty so that tags can be read in an objective way.

In this article, we will show how POWFs can be employed in situations where complete transparency is required at the system level while providing all the requisite features at the tag level.

3 The physics of POWFs

A typical POWF embodiment is a token (e.g., a credit card, access control fob) encapsulating a small, optically translucent three-dimensional microstructure which contains inhomogeneities (also referred to as scatterers) that have features at the scale of the wavelength of visible light. The token is probed using a laser beam as shown in Figure 1 below. The scattering of such coherent radiation from an inhomogeneous medium produces laser speckle fluctuations, which is the result of interference of light that has taken a multitude of paths through the token. This speckle pattern is a complicated function of the microstructure of the

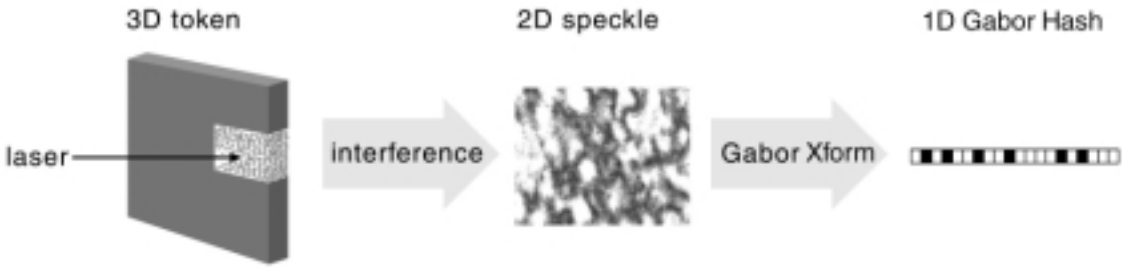

Figure 1: Coherent multiple scattering from an inhomogeneous structure results in a laser speckle pattern that can be reduced to a binary string. This string can be used as a unique identifier for the structure. This process may be viewed as physically hashing the complicated 3D structure down to a fixed-length key. The Gabor Transform is a means of filtering noisy speckle patterns and reducing them to a fixed-length bitstring called a Gabor Hash. Both these terms are defined in the text.

token and is used to derive a unique identifier for the structure.

We will show in the rest of this article that

- each token can produce not one but a very large number of identifiers. The availability of a large number of identifiers allows its deployment in challenge-response protocols.
- under certain conditions, each of these identifiers is a string of random bits.
- making small changes to the token's structure causes a given identifier to completely decorrelate

Before we press on into the physics of POWFs, consider a general model for the underlying physical mechanism. Typically, we have a physical system S encapsulated in a token and a physical probe P that interacts with S to produce an output O which is recorded by a detector D (Figure 2). How can we build a system that allows us to repeatably and robustly distinguish S from others in its class? Clearly, there are several choices for each of the elements in the system: S could be drawn from a large number of physical systems (e.g., regular vs. disordered, 2D vs. 3D); the probe P could possess several attributes (electromagnetic vs. acoustic, single frequency vs. broadband); and the detector could be anything from a voltmeter to a digital camera to X-ray film depending on the nature of S and P.

The long list of candidate systems, probes, and detectors may be narrowed down by considering the crucial properties they must have in order to meet our requirements.

- *Uniqueness* requires that the output O as recorded by D have a large number of statistically independent degrees of freedom
- *Tamper resistance* requires that the output O have a sensitive dependence on the state of S
- *Unforgeability* requires that the system S be difficult and expensive to clone regardless of the prior knowledge a forger has of P and O

Mesoscopic systems are a large class of physical systems that possesses all these properties. They are so named because they lie in a region between *macroscopic systems*, which are governed by the laws of classical physics, and *microscopic systems*, which

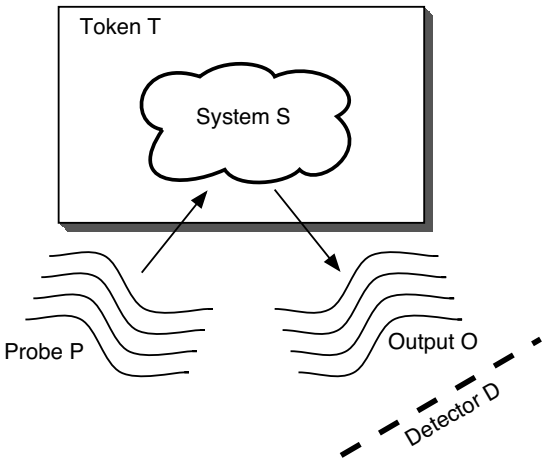

Figure 2: A general model for a physical authentication system

are governed by *quantum physics*. The fundamental distinguishing feature of mesoscopic systems is the preservation of coherence as radiation travels through the system i.e., the wavelength of the radiation is unchanged and its phase relative to that of the incident radiation is predictable after it exits the system. When *disordered* mesoscopic systems are probed with coherent radiation, the interference pattern after the radiation has passed through the structure is called a speckle pattern or a conductance fluctuation [13, 22]. By contrast, ordered mesoscopic systems produce regular diffraction patterns which are easily predictable given knowledge of the structure and the probe. In fact it is possible to predict the structural configuration by observing the diffraction patterns, a fact that is commonly used in X-ray crystallography. Hereafter, we will focus our attention on disordered mesoscopic systems.

There are generally four length scales of importance in these systems. The first is the wavelength λ of the incident probe. The second is the mean free path l which is the average distance between scattering events within the physical system S. The third is the size of the physical system itself denoted by L and finally, we have the coherence length[1] of the probe radiation L_c. The mesoscopic regime is governed by the inequality $\lambda \ll l \ll L \ll L_c$. In the mesoscopic limit of scattering in a three-dimensional structure [7, 8], the mean free path l between elastic collisions with scatterers is much larger than the wavelength λ of the radiation, but the thickness L of the structure is much smaller than the coherence length of the probe. In this regime of *coherent multiple scattering*, if the cross-sectional area of a beam is A, then moving $A/(Ll)$ scatterers will produce an uncorrelated speckle pattern, as will rotating the incident beam by an angle $\delta\theta = \lambda/(2\pi L)$ [2]. This phenomenon is the physical basis for POWFs.

We have thus narrowed down the choices of system components to:

- *Physical system S* - a 3D structure in the mesoscopic regime i.e., whose size L lies between the wavelength of the probe radiation and the coherence length of the radiation. This structure contains numerous scatterers which have features at the scale of the wavelength of the physical probe.
- *Physical probe P* - coherent radiation at a given wavelength λ
- *Interaction mechanism* between the *P* and *S* is coherent multiple scattering i.e., the interaction of coherent radiation with multiple scatterers in the disordered microstructure

The choice of detector D depends intimately on the wavelength of radiation. We note that although mesoscopic behavior is observed at all wavelengths, our requirement for unforgeability places an upper bound on the wavelength. Specifically, the size (in units of λ) of the structure L and its disorder characterized by the mean free path l must be such that it is prohibitive to

[1]The coherence length is defined as the distance light is able to travel from the laser before its phase becomes unpredictable relative to that at the laser.

clone the disordered mesoscopic system. Practically speaking, given the state of the art in 3D microfabrication, this restricts the range of wavelengths to be below one micron.

4 Implementation

In the embodiment described here [20, 21], we used a $\lambda = 632.8$ nm HeNe laser beam to illuminate $10 \times 10 \times 2.5$ mm^3 optical epoxy tokens containing 500-800 μm glass spheres. This represents about a penny's worth of materials; the cost of the reader can be anything from a few dollars to several hundreds of dollars depending on the precision of the laser pointing system. The density of spheres was chosen to give an average spacing on the order of 100 μm, which equals the photon mean free path in the limit of strong scattering applicable here [22]. The resulting speckle patterns were recorded with an inexpensive 320×240 pixel CCD camera. Repeatable positioning, i.e., mechanical registration, of the token with respect to the probe and the detector is achievable without recourse to high-precision (and expensive) systems.

Although it is possible to use the speckle patterns directly as identifiers, this is error-prone owing to the noisy readings of speckle patterns and their inherent sensitivity to small changes in the state of the probe. Figure 3 provides an example of the noise that can occur in a POWF system. In order to reduce the effects of noise, we transform the speckle pattern into a bit string using a multiscale Gabor Transform [11, 6]. The Gabor Transform is a complex-valued transform that represents speckle image intensity as a linear combination of oriented, modulated Gaussian filters at multiple scales. The parameters of the transform [19] are dependent on geometry of the specific optical implementation [20] and were experimentally determined in the embodiment discussed in this article.

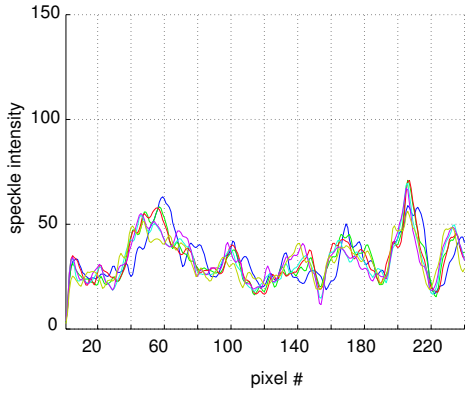

Figure 3: Noise sources in a speckle image. The plot shows six overlaid traces of speckle image intensity taken along a single row of a 320×240 raw speckle image before it was Gabor-transformed. Each trace was obtained after the token was removed and replaced in the reader after routine handling. First, there is pixel-scale noise, which is either due to the optical system or induced by the CCD detector. Then, there is noise at the scale of several pixels, which occurs at the physical interaction level. A third source of noise is due to misregistration of the token, shown as a trace horizontally displaced from the rest by about 10 pixels. Another source of noise, not shown in the figure, is due to changes in average illumination levels which would manifest itself as a vertical displacement of one trace from the other.

To summarize what we have said so far, a single probe of the 3D microstructure results in a 320×240 pixel speckle intensity image that is reduced to a bitstring of length 2400. This string is the unique identifier of the 3D microstructure when interrogated with a probe beam in a given state. Hereafter, we will refer to this bitstring as a *Gabor Hash*.

5 Experiments

In this section we present several experimental results that elucidate the properties of POWFs. The first

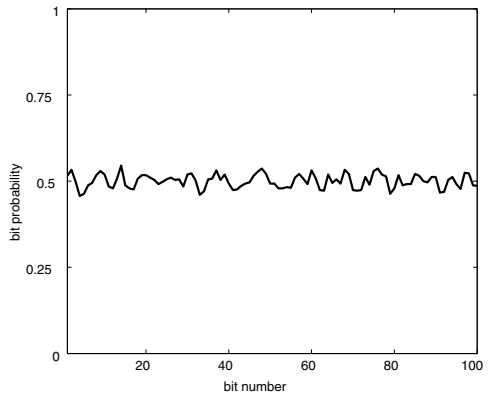

Figure 4: The plot depicts the probability that a bit in any given location of a Gabor Hash is set or cleared. Although only a 100-bit window is shown for clarity, this behavior is observed over all the bits.

experiment explores the average probability that a bit in any given location is either *0* or *1*. The average is taken over 576 Gabor Hashes which were derived from four different tokens, each of which was probed at 144 distinct locations. Figure 4 plots this probability, which hovers around 0.5. This result clearly indicates that each bit is equally likely to be set or cleared i.e., the bitstring derived from a POWF is a bit-wise maximum entropy code.

The second experiment focuses on how effective the Gabor Hash is at distinguishing one token from another. To ascertain this, we used the Gabor Hashes gathered in the previous experiment in an enrollment/authentication scenario. The bitstrings were enrolled in a database and candidate bitstrings were (a) matched to corresponding enrolled bitstrings and (b) matched to all 575 non-corresponding enrolled bitstings. The metric used for matching was a normalized Hamming Distance (i.e, every bit being different equals a distance of 1). The *like distribution*, which is the Hamming Distance distribution obtained from matching Gabor Hashes which had the same origin and the *unlike distribution* obtained by matching Gabor Hashes which had distinct origins are shown in Figure 5.

We learn several facts about POWFs from Figure 5. The distance between Gabor Hashes that have the same origin is usually smaller than the distance between hashes that have different origins. The average Hamming Distance between Gabor Hashes that have different origins is 0.5 implying that one can do no better at guessing one from the other than coin-flipping. The fact that the like distribution may be modeled by a binomial distribution with 233 independent degrees of freedom implies that this implementation of POWFs is capable of distinguishing between $2^{233} \approx 10^{70}$ Gabor Hashes. However, only a small subset of these Gabor Hashes are available from the same token. The number of available Gabor Hashes from any given token is calculated below.

From theory, we know that moving the probe beam by a small angle or displacing a small number of scatterers (see section 3) causes the speckle pattern to decorrelate completely. For our implementation the theoretically calculated value of angular displacement of the probe beam required to cause decorrelation of the speckle pattern is $\delta\theta = \lambda/(2\pi L) = 4 \times 10^{-5}$ rad. In practice, this value is ~ 40 times greater and equal to 1.7×10^{-3} rad. Linear sensitivity is challenging to calculate theoretically, but was experimentally determined to be 60μm. These results place constraints on the mechanical system that must be used to register the tokens with respect to the probe beam. Figure 6 shows the linear and angular sensitivity plots. For our 100 mm^2 token, assuming that the range of possible probe angles are bounded by $\Delta\theta = \pi/2$, we have a total of $2.37 \times 10^{10} \approx 2^{34}$ available Gabor Hashes from any given token. This number may be made larger by using higher precision probe positioning equipment. Obviously, using such equipment would increase the cost of the reader substantially.

One final point to note is the crossover point between distributions in Figure 5. The two distributions intersect at a Hamming Distance of 0.41. This means that up to $2400 * 0.41 = 984$ bits can be wrong in a given Gabor Hash before we reject it as being unre-

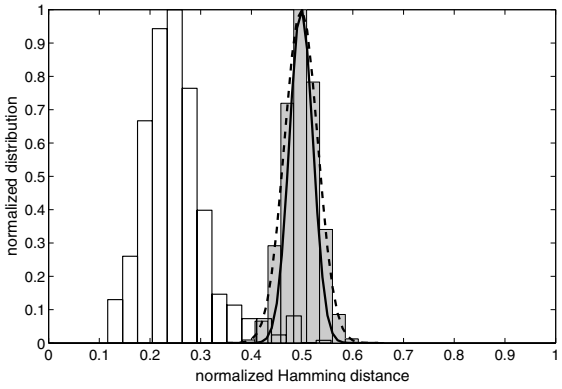

Figure 5: The normalized Hamming distances measured for Gabor Hashes. The unlike distribution, in gray, shows the distribution of 165,600 distances between unlike bitstrings; the mean of the dashed Gaussian fit is 0.50 - half the bits differ on average - and the variance is 1.07×10^{-3} (equivalent to 233 independent binomial trials). Doubling the length of the bitstring to 4800 bits by concatenating readings from two angles produces a distribution with a Gaussian fit shown by the solid curve, reducing the variance to 5.42×10^{-4}, corresponding to 461 independent binomial trials. The like distribution, in white, shows the errors in rereading 576 like bitstrings after candidates are presented to the enrolled database; the mean of 0.25 equals 1800 bits being matched correctly.

lated to a previously enrolled one. While this amount of noise tolerance is essential to keep the cost of the readers low, it offers increased probability of successful spoofing for an attacker. We will have more to say about this later. The final experiment demonstrates tamper-resistance. One Gabor Hash was enrolled in a database, and a second one was obtained from the same token after it was intentionally damaged by drilling a 1 mm deep hole in its surface with a drill of diameter 533μm. The distance between the two hashes was 0.46, thereby physically demonstrating avalanche. Figure 7 shows the results of this experiment.

Thus far, we have experimentally characterized both uniqueness - a large number of independent degrees of freedom in the Gabor Hash - and tamper-resistance - a sensitive dependence on the state of the system and the probe - in our embodiment of POWFs. We leave the discussion of unforgeability to a later section.

6 Abstraction

From a cryptographic point of view, it is useful to model POWFs as follows [17]. A (k, n)-POWF Π comprises a set of values $\{\Pi(i)\}_{i=1}^{n}$, where each $\Pi(i) \in \{0,1\}^k$ is generated independently and uniformly at random. Π may be conceptualized as a tape consisting of n cells, each of which contains a k-bit string. The value n will in general be finite in a POWF, as a reflection of practical limitations on the number of possible ways in which the underlying physical object may be read. This representation assumes that it is possible to go from a 2400-bit Gabor Hash with correlations between bits to a shorter 233-bit sequence of uncorrelated bits. This may be accomplished by one of the many available methods of entropy coding [5].

In response to a challenge $i \in \{1, 2, \ldots, n\}$ to POWF Π, the response $\Pi(i)$ is returned. This value, however, is communicated through a noisy channel ν. In other words, the response received by the challenger is a random variable $\Pi_\nu(i)$ over the space $\{0,1\}^k$ that models the effects of various types of noise on the underlying value $\Pi(i)$.

The POWF we consider has $k = 2400$ bits, although this number would go down to 233 if some form of entropy coding were used to compress a Gabor Hash. As we saw above, the number of cells (i.e., unique challenges) supported by any token is $n \approx 10^{10} \approx 2^{34}$. We note that k may be increased in practice by (a) reducing the noise in the system through better engineering and (b) using a larger detector. n can be increased by increasing the size L of the structure, decreasing the mean free path l be-

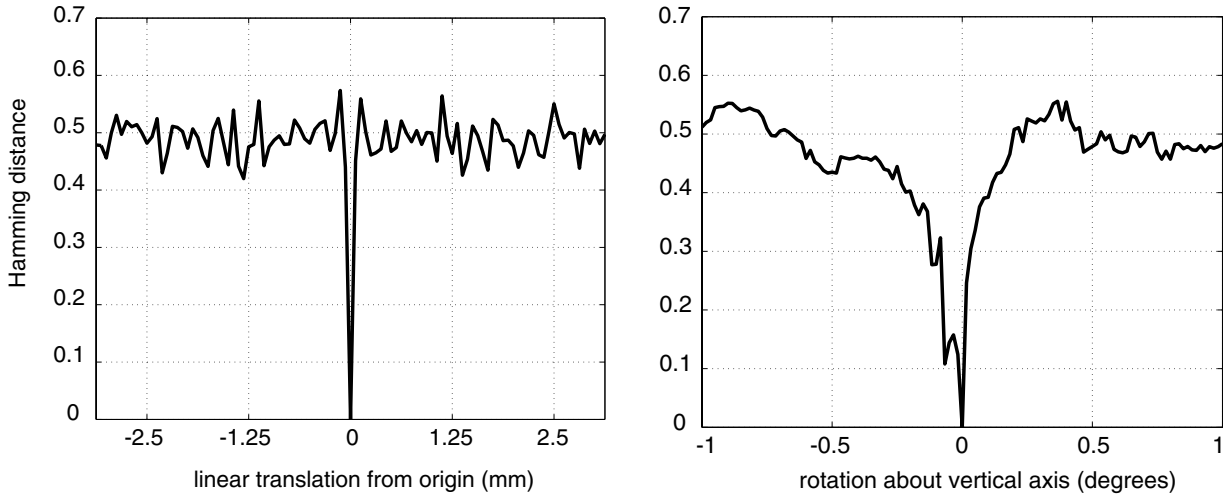

Figure 6: The plot on the left shows the Hamming distance between a reference key obtained from a central location and keys obtained as the laser is translated linearly across the surface of the token. A translation of approximately 60 microns causes the key to decorrelate completely. Data obtained for angular sensitivity show that a rotation of approximately 1.7 mrad causes full decorrelation of the key.

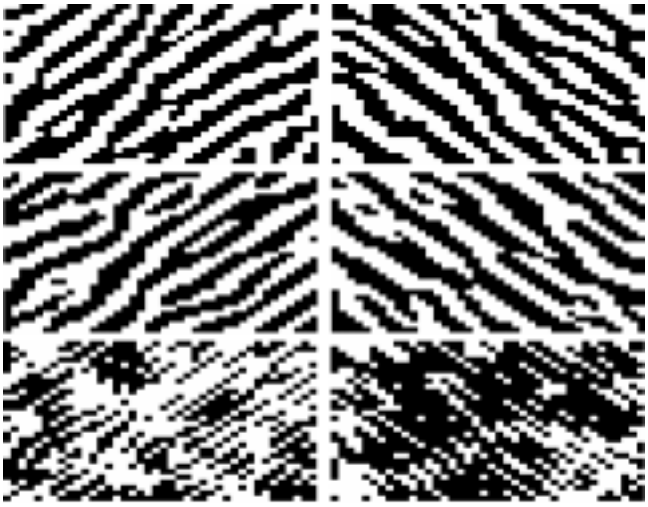

Figure 7: Demonstration of tamper resistance. The top row shows a segment of the enrolled Gabor Hash (represented as two adjacent binary images for ease of visualization), the middle row shows the same segment from the hash of an intentionally damaged token, and the bottom row shows the XOR of the previous two rows. The Hamming Distance between the top two rows is 0.46.

tween scatterers up to a limit equal to the wavelength of probe radiation, decreasing the wavelength λ of the probe radiation, and engineering higher precision probe positioning systems. One final point to note is that, from an economic point of view, increasing the size of the structure or decreasing the mean free path adds very little cost, if any, to the system.

7 Attacks

POWFs, as described here, may be used in an authentication protocol as described in Figure 8. The protocol is based on generating challenge-response pairs on secure terminals and consuming them on unsecure terminals. During the enrollment stage, several challenge-response pairs (denoted by (θ, k)) are acquired at a trusted terminal. During the verification stage of the protocol, the server challenges the token with a specific θ_i and compares the noisy response k_i' with the known k_i. The token is authenticated if the Hamming distance between k_i' and k_i is below a previously set threshold T. The challenge-response pair

(θ_i, k_i), grayed out in Figure 8, is not reused in any future transactions. As we saw earlier, the number of challenges per token is $\approx 2^{34}$. Given that up to 41% of the bits can be incorrect before we reject a spoofed Gabor Hash as having not originated in the same token, the probability that an attacker can guess the corresponding response is $2^{-233*0.59} \approx 2^{-137}$.

An attack on a POWF-based authentication system is successful if an attacker can demonstrate possession of the POWF without actually having physical access to the token. Stated more formally, we would like to enumerate the subset of cells of a (k, n)-POWF an attacker can spoof when challenged with queries $i \in \{1, 2, \ldots, n\}$. There are two classes of attacks on POWFs - physical and computational - each offering varying degrees of ease of spoofing to the attacker. Physical attacks are of interest in environments where the reader requires the presence of a 3D structure as part of the authentication process while computational attacks are relevant in scenarios where the Gabor Hashes, rather than the 3D structure itself, are used.

7.1 Physical attacks

These attacks involve creating a physical structure that emulates all or part of a (k, n)-POWF. The spectrum of attacks ranges from a static image that spoofs a single cell to holograms that spoof a small subset of cells to cloning the entire structure down to the scale of λ. The former attack may be thwarted by using the POWF in a challenge-response protocol as described in Figure 8. The holographic attack is practically infeasible owing to limitations in the ability of holographic film to store and reproduce a large number of images with no crosstalk [16]. The most difficult attack of all is the cloning attack. The principal difference between that holographic attack and the cloning attack is that the hologram aims to emulate the optical behavior of the 3D microstructure without actually creating a replica of the structure itself. The state of the art in 3D microfacrication is far behind the difficulty presented by a macroscopic 3D structure with λ-scale inhomogeneities [18]. This difficulty is further enhanced because probe samples not just the token's physical structure but also its material properties (e.g. dielectric constant) of the medium as well as those of the scatterers. This implies that in order for a cloning attack to succeed, it would have to not only recreate the structure but also its local electromagnetic attributes. Given the fact that 3D microfabrication is currently possible with only a small library of materials, it appears that a full-fledged cloning attack is infeasible using know 3D microfacrication technology. Finally, we remark that spoofing a single token does not affect the integrity of any other token.

7.2 Computational attacks

This class of attacks involves spoofing the (k, n)-POWF computationally. The simplest of these attacks is a replay attack - observe and store all possible challenges and corresponding responses for replay later. This attack is the most feasible of all and involves storage of a large amount of data. For a spoofing success probability of 100%, our (2400, $\sim 10^{10}$)-POWF requires storing 2400×2^{34} bits $\approx 2^{45}$ bits i.e., about 32 terabytes of data, which is not infeasible, but it *is* expensive. If the attacker were satisfied with a lower success rate, the storage requirements would decrease accordingly. This decrease in storage could be offset by requiring the verifying server to challenge the prover multiple times. Storage requirements drop to about 4 terabytes if the Gabor Hashes were compressed to 233 bits. Further, assuming a 10 ms acquisition time per response, it would take an attacker over 3 years to acquire responses to all possible challenges. Note that the verifier's database can be much smaller because it can select the subset of challenge-response pairs that it wants to query on in advance.

A second attack would be simulate the response to any given challenge. Assume that the volume of the token is 1 cm^3 and it is probed by light with a wave-

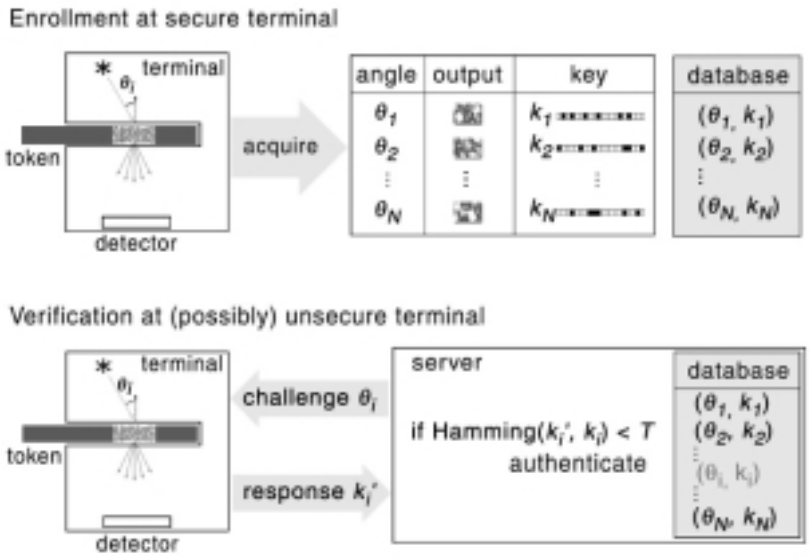

Figure 8: A challenge-response authentication protocol

length on the order of 1 μm, then its structure is specified by up to $(10^{-2}/10^{-6})^3 = 10^{12} \approx 2^{40}$ bits if the composition of each cubic block of wavelength size is random, as it would be for microscopically inhomogeneous scatterers. These bits could be used to computationally simulate the output instead of storing all possible outputs in advance. In the mesoscopic limit, a photon passing through the structure performs a random walk with a step size given by the mean free path l, covering a distance $l\sqrt{N}$ after N scattering events [23]. For the photon to emerge from the thickness L of the token requires that $L = l\sqrt{N}$ and so $N = (L/l)^2$ scattering events. At each of these steps, in a simulation it is necessary to propagate forward paths linking all pairs of scatterers, giving an total of $\sim 10^{12} \times 10^{12} \times 10^2 = 10^{26} \approx 2^{86}$ scattering simulations per scattering event. In our embodiment, $N = 625$ scattering events. In practice, simulating the scattering from even a single arbitrarily-shaped particle in the limit that its dimension is several times the wavelength presently requires a supercomputer [15]. Although simulating the response to any arbitrary challenge is not provably difficult, it does require complete knowledge of local structural and electromagnetic properties of the microstructure at the scale of λ and access to extremely high-performance computing. This presents a substantial challenge to any attacker.

Successfully spoofing a POWF involves technical measures, effort and expense which are extremely disproportional to the effort and expense of creating the POWF. This physical asymmetry is akin to the computational asymmetry encountered in cryptographic one-way functions.

8 Discussion

POWFs are expected to find utility in physical authentication systems where challenge-response protocols are employed. A typical application could be in access control, where the number of tokens is small and the data system employed usually relies on a

trusted central computer to keep track of the challenges and responses. Another potential application is in arms control treaty verification. Unlike more familiar challenge-response protocols, this one relies on the enormous amount of information that is committed in advance to the token that is read out over a long period. Beyond this, applications exist in tamper resistant packaging either as externally monitored free-standing structures or through the use of self-contained packages containing a laser, a detector which are potted in optical epoxy containing inhomogeneities. It is also worth noting that POWFs can be built at any wavelegth as long as the system is in the mesoscopic regime. Speckle patterns have been observed mesoscopic all-electronic systems by using the scattering of electrons from atomic-scale inhomogeneities [14]. Although the temperature at which these effects are observed is too low to be practically used, this line of thinking opens up new approaches to uniquely identifying electronic structures based solely on their physical structures. One area where this kind of identification is becoming increasingly important is in assigning identity to silicon chips. Recent work in silicon POWFs, Physical Unknown Functions (PUFs) and Physical Random Functions (PRFs) [3, 12] points to interesting opportunities in using the actual physical structure of silicon chips for identification, certified execution, and digital rights management.

We have shown how coherent multiple scattering in inexpensive 3D structures performs a mapping that satisfies all of the attributes of a physical source of data with properties akin to those of a noisy random oracle [17]. The value of POWFs lie in the fact that they, unlike prior physical authentication methods, makes an explicit connection with the framework of modern cryptography and thus may be viewed as another primitive in the cryptographer's toolbox, albeit one that has a physical manifestation. In cases where cryptographic authentication is neither economically nor practically feasible, POWFs offer an alternative approach.

Acknowledgements

The author thanks Ari Juels for several insightful discussions. Markus Jakobsson, Burt Kaliski, and Ari Juels also provided a large number of comments during review which greatly improved the content and readability of this article.

References

[1] C. Bennet, G. Brassard, S. Breidbard, and S. Wiesner. Quantum cryptography, or unforgeable subway tokens. In *Proceedings of Crypto '82*, pages 267–275, 1982.

[2] R. Berkovits. Sensitivity of the multiple-scattering speckle pattern to the motion of a single scatterer. *Physical Review B*, 43:8638–40, 1991.

[3] D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. Secure hardware processors using silicon physical one-way functions. In R. Sandu, editor, *ACM CCS '02*, 2002.

[4] D. Collon. *First Impressions: Cylinder seals in the Ancient Near East*. British Museum, London, 1987.

[5] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, New York, 1991.

[6] J. G. Daugman. Uncertainty relation for space, spatial frequency, and orientation optimized by two dimensional visual cortical filters. *Journal of the Optical Society of America*, 2:1160–9, 1985.

[7] S. Feng, C. Kane, P.A. Lee, and A.D. Stone. Correlations and fluctuations of coherent wave transmission through disordered media. *Physical Review Letters*, 61:834–7, 1988.

[8] S. Feng and P.A. Lee. Mesoscopic conductors and correlations in laser speckle patterns. *Science*, 251:633–9, 1991.

[9] Steve Fetter and Thomas Garwin. Using tags to moniitor numerical limits in arms control agreements. In Barry M. Blechman, editor, *Technology and the Limitation of International Conflict*, pages 33–54, Washington, DC, 1989. The John's Hopkins Foreign Policy Institute.

[10] Steve Fetter and Thomas Garwin. Tags. In Richard Kokoski and Sergey Koulik, editors, *Verification of Conventional Arms Control In Europe: Technological Constraints and Opportunities*, pages 139–154, Boulder, CO, 1990. Westview Press.

[11] D. Gabor. Theory of communication. *Journal of the Institute of Electrical Engineers*, 93:429–457, 1946.

[12] B. Gassend. Physical random functions. Master's thesis, Massachusetts Institute of Technology, 2003.

[13] J.W. Goodman. Statistical properties of laser speckle patterns. In J.C. Dainty, editor, *Laser Speckle and Related Phenomena*, pages 9–75, Berlin, 1975. Springer-Verlag.

[14] D. Hoadley, P. McConville, O. Norman, and N.O. Birge. Experimental comparison of the phase-breaking lengths in weak localization and universal conductance fluctuations. *Physical Review B*, 60:5617–25, 1999.

[15] A.G. Hoekstra, M.D. Grimminck, and P.M.A. Sloot. Large scale simulations of elastic light scattering by a fast discrete dipole adapproximation. *International Journal of Modern Physics C*, 9:87–102, 1998.

[16] K.M. Johnson, L. Hesselink, and J.W. Goodman. Holographic reciprocity law failure. *Appl. Optics*, 23:218–227, 1984.

[17] A. Juels and R. Pappu. Physical random oracles, 2003. manuscript in preparation.

[18] C. Marxer and N.E. de Rooij. Silicon micromechanics for the fiber-optic information highway. *Sensors and Materials*, 10:351–62, 1998.

[19] O. Nestares, R. Navarro, J. Portilla, and A. Tabernero. Efficient spatial-domain implementation of a multiscale image representation based on gabor functions. *Journal of Electronic Imaging*, 7:166–73, 1998.

[20] R. Pappu. *Physical One-Way Functions*. PhD thesis, Massachusetts Institute of Technology, 2003.

[21] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. *Science*, 297:2026–2030, 2002.

[22] M.C.W. van Rossum and Th.M. Nieuwenhuizen. Multiple scattering of classical waves: Microscopy, mesoscopy, and diffusion. *Reviews of Modern Physics*, 71:313–371, 1999.

[23] Bart van Tiggelen. *Multiple Scattering and Localization of Light*. PhD thesis, University of Amsterdam, 1992.

[24] W. K. Wootters and W. Zurek. A single quantum cannot be cloned. *Nature*, 299:982–83, 1982.

[25] J. Yoshida. Euro bank notes to embed RFID chips by 2005. *EE Times*. 19 December 2001. Available at www.eetimes.com/story/OEG20011219S0016.

ABOUT RSA LABORATORIES

An academic environment within a commercial organization, RSA Laboratories is the research center of RSA Security Inc., the company founded by the inventors of the RSA public-key cryptosystem. Through its research program, standards development, and educational activities, RSA Laboratories provides state-of-the-art expertise in cryptography and security technology for the benefit of RSA Security and its customers.

Please see www.rsasecurity.com/rsalabs for more information.

NEWSLETTER AVAILABILITY AND CONTACT INFORMATION

CryptoBytes is a free publication and all issues, both current and previous, are available at www.rsasecurity.com/rsalabs/cryptobytes. While print copies may occasionally be distributed, publication is primarily electronic.

For more information, please contact:

cryptobytes-editor@rsasecurity.com.

RSA SECURITY®

RSA Security Inc.
www.rsasecurity.com

RSA Security Ireland Limited
www.rsasecurity.ie