A Threshold Pseudorandom Function Construction and Its Applications

Jesper Buus Nielsen*

BRICS** Department of Computer Science
University of Aarhus
Ny Munkegade
DK-8000 Arhus C, Denmark

Abstract. We give the first construction of a practical threshold pseudorandom function. The protocol for evaluating the function is efficient enough that it can be used to replace random oracles in some protocols relying on such oracles. In particular, we show how to transform the efficient cryptographically secure Byzantine agreement protocol by Cachin, Kursawe and Shoup for the random oracle model into a cryptographically secure protocol for the complexity theoretic model without loosing efficiency or resilience, thereby constructing an efficient and optimally resilient Byzantine agreement protocol for the complexity theoretic model.

1 Introduction

The notion of pseudorandom function was introduced by Goldreich, Goldwasser and Micali[GGM86] and has found innumerable applications. A pseudorandom function family is a function F taking as input a key K and element x, we write $F_K(x)$, where for a random key the output of F_K cannot be distinguished from uniformly random values if one does not know the key. If one have to require that the input of F_K is uniformly random for the output of F_K to look uniformly random, we say that F is weak pseudorandom.

One immediate application of pseudorandom functions is using them for implementing random oracles: Consider a protocol setting with n parties. A c-threshold random oracle with domain D is an ideal functionality (or trusted party). After c parties have input $(\texttt{evaluate}, x)$, where say $x \in \{0,1\}^*$, the functionality will return a uniformly random value $r_x \stackrel{R}{\leftarrow} D$ to all parties that input $(\texttt{evaluate}, x)$. This functionality defines a uniformly random function from $\{0,1\}^*$ to D. Numerous protocol constructions are known that can be proved secure assuming that a random oracle is available. However, any implementation of such a protocol must also provide an implementation of the oracle. In practice, a hash function is often used to replace a 1-random oracle, but then the implementation is only secure if an adversary can do no better with the hash

* `buus@brics.dk`.
** Basic Research in Computer Science,
Centre of the Danish National Research Foundation.

function than he could with the oracle. This is something that in general cannot be proved, but must be a belief based on heuristics — in fact, for some protocols, this belief is always wrong [CGH98,Nie02].

In contrast, pseudorandom functions can be used to implement random oracles without loss of security. This can be done by generating K at the beginning of the protocol and letting $r_x = F_K(x)$ when r_x is needed. It is however clearly necessary that no party should know the key of F_K, since the output of a pseudorandom function only looks random to parties who do not have the key. Therefore the key — and hence also ability to evaluate the function — must be distributed among the parties using, for instance, a threshold secret-sharing scheme.

Our Results In this paper we construct a new pseudorandom function family. The key will be a prime Q, where $P = 2Q + 1$ is also a prime, a random value x from the subgroup Q_P of $\boldsymbol{Z}_P^*$ of order Q, along with $2l$ random values $\{\alpha_{j,b}\}_{j=1,\ldots,l,b=0,1}$ from $\boldsymbol{Z}_Q$. The function family maps from the set of strings of length at most l to Q_P, and given $\sigma = (\sigma_1, \ldots, \sigma_m) \in \{0,1\}^{\leq l}$, the output will be $x^{\prod_{i=1}^{m} \alpha_{i,\sigma_i}} \bmod P$. We prove this function family secure under the decisional Diffie-Hellman (DDH) assumption.

More importantly, we give a secure n-party protocol for evaluating the function. Our protocol is for the asynchronous model with authenticated public point-to-point channels. This is a very realistic model of communication and can be efficiently implemented[CHH00]. The protocol is statically secure as long as less than $n/3$ parties misbehave. In some applications the protocol can communicate as much as $O(ln^2k)$ bits per evaluation, where k is the security parameter (for each exponent each party sends to each other party k bits). However, in many uses the communication complexity will be $O(n^2k)$ bits.

To demonstrate the applicability of our new threshold pseudorandom function, we describe how to implement efficient Byzantine agreement (BA) in the complexity theoretic model, by replacing the random oracles in the protocol [CKS00] with our threshold pseudorandom function. The resulting protocol has the same resilience as the protocol in [CKS00], namely resilience against a malicious coalition of one third of the parties. It has the same communication complexity of $O(n^2k)$ bits per activation and the same (constant) round complexity up to a small constant factor. As part of the implementation we show how to replace the random oracles in the threshold signature scheme from [Sho00] by our threshold pseudorandom function.

Related Work The notion of distributed pseudorandom function, which is similar to our threshold pseudorandom function, was introduced by Naor, Pinkas and Reingold in [NPR99]. They do not define distributed pseudorandom functions in a general multiparty computation model — their model is more ad-hoc and implementation-near. Since there are differences between the two notions, we have chosen a different name for our definition.

Until now the most efficient known construction of threshold pseudorandom functions was using general multiparty computation techniques or coin-toss protocols, or were restricted to a (logarithmic) small number of parties because of the way the key was distributed[MS95,BCF00].

In [NPR99] an efficient threshold *weak* pseudorandom function was constructed based on the DDH assumption, and it was left as an interesting open problem to construct an efficient threshold pseudorandom function. Our protocol uses the protocol from [NPR99]. Indeed, our construction contains a general and efficient construction of threshold pseudorandom functions from threshold weak pseudorandom functions. This technique is reminiscent of the construction of pseudorandom functions from pseudorandom generators in [GGM86].

Our pseudorandom function is similar to a function from [NR97], but there are some essential differences, which allows to efficiently distribute our function. Indeed, the construction from [NR97] does not seem to allow an efficient secure distributed protocol.

Organization In Section 2 we give some preliminary notation and definitions. In Section 3 we describe our pseudorandom function and prove that it is pseudorandom under the DDH assumption. In Section 4 we sketch the framework for secure multiparty computation from [Can01] and define the notions of threshold function family, c-threshold random oracle, threshold trapdoor permutation, threshold signatures and Byzantine agreement in this framework. In Section 5 we construct a threshold pseudorandom function by giving a distributed protocol for our threshold pseudorandom function. In Section 6 we show how to use this threshold function family and the RSA based threshold trapdoor permutation from [Sho00] to implement a threshold signature scheme based on the RSA and DDH assumptions. Finally in Section 7 we show how to use this threshold signature scheme along with our threshold pseudo-random function to implement the BA protocol from [CKS00] in the complexity theoretic model under the RSA and DDH assumptions.

2 Preliminaries

We use ϵ to denote the empty string and for $l \in \boldsymbol{N}$ we use $\{0,1\}^{\leq l}$ to denote the set $\bigcup_{i=0}^{l}\{0,1\}^i$ of all strings of length at most l. For a set S we use $x \stackrel{R}{\leftarrow} S$ to denote the action of sampling an element x (statistically close to) uniformly random from S, and for a probabilistic algorithm we use $a \stackrel{R}{\leftarrow} A$ to denote the action of running A with uniformly random bits and letting a be the output. We use $k \in \boldsymbol{N}$ to denote the security parameter. We will often skip the security parameter if it is implicitly given by the context. If e.g. $S = \{S_k\}_{k\in\boldsymbol{N}}$ is a sequence of sets we will write $x \in S$ to mean $x \in S_k$, where k is the security parameter given by the context.

Trapdoor Commitment Scheme A trapdoor commitment scheme can be described as follows: first a public key pk is chosen based on a security parameter k, by running a probabilistic polynomial time (PPT) *generator* G. Further more, there is a fixed function $\mathtt{commit}$ that the committer C can use to compute a commitment c to s by choosing some random input r, computing $c = \mathtt{commit}_{pk}(s, r)$, and sending c. Opening takes place by sending (s, r); it can then be checked that $\mathtt{commit}_{pk}(s, r)$ is the value S sent originally. We require that the scheme is perfect hiding and computationally binding. The algorithm for generating pk also outputs a string t, the trapdoor, and there is an efficient algorithm which on input t, pk outputs a commitment c, and then on input any s produces uniformly random r for which $c = \mathtt{commit}_{pk}(s, r)$. In other words, a trapdoor commitment scheme is binding if you know only the public key, but given the trapdoor, you can cheat arbitrarily and undetectable.

Pseudorandom Functions The following definitions are adaptions of definitions from [Gol01,BDJR97].

Definition 1. *A* function family *is a sequence $F = \{F_k\}_{k \in \boldsymbol{N}}$ of random variables, so that the random variable F_k assume values which are functions. We say that a function family is a* PPT function family *if the following two conditions hold:*

Efficient indexing *There exists a PPT algorithm, I, and a mapping from strings to functions, ϕ, so that $\phi(I(1^k))$ and F_k are identically distributed. We denote by f_i the function $\phi(i)$.*

Efficient evaluation *There exists a PPT algorithm, V, so that $V(i, x) = f_i(x)$.*

Let F and G be two function families. We write $F \subset G$ if for all k the functions that receive non-zero probability mass in F_k is a subset of the functions that receive non-zero probability mass in G_k. Consider two sequences $A = \{A_k\}_{k \in \boldsymbol{N}}$ and $B = \{B_k\}_{k \in \boldsymbol{N}}$ of sets. If all A_k and B_k are finite, we use $[A \to B]$ to denote the function family $\{F_k\}_{k \in \boldsymbol{N}}$ where F_k is uniform over the set of all functions from A_k to B_k. We say that a function family F maps from A to B if $F \subset [A \to B]$.

Definition 2. *Let $F \subset [A \to B]$ be a PPT function family. Let $b \in \{0, 1\}$. Let D be a distinguisher that has access to an oracle. Let $\mathcal{O}_f$ be the oracle which on input $s \in A$ outputs $f(s)$, and let $\mathcal{R}_f$ be the oracle which on input* $\mathtt{gen}$ *generates a uniformly random $s \in A$ and outputs $(s, f(s))$. Now consider the following experiments and corresponding advantages.*

$$
\begin{array}{ll}
\underline{\textbf{proc}}\ \textbf{Exp}_{F,D}^{\texttt{wprf-b}} \equiv & \qquad \underline{\textbf{proc}}\ \textbf{Exp}_{F,D}^{\texttt{prf-b}} \equiv \\
\quad f_0 \stackrel{R}{\leftarrow} [A \to B] & \qquad \quad f_0 \stackrel{R}{\leftarrow} [A \to B] \\
\quad f_1 \stackrel{R}{\leftarrow} F & \qquad \quad f_1 \stackrel{R}{\leftarrow} F \\
\quad d \leftarrow D^{\mathcal{R}_{f_b}} & \qquad \quad d \leftarrow D^{\mathcal{O}_{f_b}} \\
\quad \underline{\textbf{return}}\ d & \qquad \quad \underline{\textbf{return}}\ d
\end{array}
$$

$$
\begin{aligned}
\boldsymbol{Adv}_{F,D}^{\mathit{wprf}} &= \Pr[\boldsymbol{Exp}_{F,D}^{\mathit{wprf\text{-}1}} = 1] - \Pr[\boldsymbol{Exp}_{F,D}^{\mathit{wprf\text{-}0}} = 1] \\
\boldsymbol{Adv}_{F,D}^{\mathit{prf}} &= \Pr[\boldsymbol{Exp}_{F,D}^{\mathit{prf\text{-}1}} = 1] - \Pr[\boldsymbol{Exp}_{F,D}^{\mathit{prf\text{-}0}} = 1]
\end{aligned}
$$

We say that F is a weak pseudorandom function family (WPRF) from A to B *(is a* pseudorandom function family (PRF) from A to B*) if for all PPT distinguishers D the advantage $\boldsymbol{Adv}^{wprf}_{F,D}$ ($\boldsymbol{Adv}^{prf}_{F,D}$) is negligible.*

3 The DDH-Tree Function Family

Definition 3. *The DDH-Tree function family is indexed by values $i = (Q, \{\alpha_{j,b}\}_{j\in\{1,\ldots,l\},b\in\{0,1\}}, x_\epsilon)$, where Q is a random k-bit prime s.t. $P = 2Q + 1$ is also a prime, l is some polynomial in k, the elements $\alpha_{j,b}$ are random in $\boldsymbol{Z}^*_Q$, and x_ϵ is random in Q_P (the sub-group of $\boldsymbol{Z}^*_P$ of order Q). For an index i we define a function $f_i : \{0,1\}^{\leq l} \to Q_P$, $f_i(\sigma) = x_\epsilon^{\prod_{i=1}^{m} \alpha_{i,\sigma_i}} \bmod P$, where $\sigma = (\sigma_1, \ldots, \sigma_m) \in \{0,1\}^{\leq l}$. We will sometimes use the notation x_σ to mean $f_i(\sigma)$, when i is clear from the context. Note that in particular, we have $f_i(\epsilon) = x_\epsilon$.*

We would like the function family to output bit-strings, instead of elements in Q_P. To this end, given an element $y \in Q_P$, let $\lfloor y \rfloor = \min(y, P-y)$. Consider then an index i as above except that Q is a $(k+\delta(k))$-bit prime, where $\log(k)/\delta(k) \in o(1)$. We define the function $g_i : \{0,1\}^{\leq l} \to \{0,1\}^k$, $g_i(\sigma) = \lfloor f_i(\sigma) \rfloor \bmod 2^k$. The DDH-Tree function family is given by the functions g_i.

Definition 4. *Given $P = 2Q + 1$ and a random generator g of Q_P, the DDH assumption is that the random variable $(g, g^\alpha \bmod P, g^\beta \bmod P, g^{\alpha\beta} \bmod P)$, where $\alpha, \beta \stackrel{R}{\leftarrow} \boldsymbol{Z}_Q$, is computationally indistinguishable from the random variable $(g, g^\alpha \bmod P, g^\beta \bmod P, g^\gamma \bmod P)$, where $\alpha, \beta, \gamma \stackrel{R}{\leftarrow} \boldsymbol{Z}_Q$.*

Theorem 1. *Under the DDH assumption, the DDH-Tree function family is pseudorandom from $\{0,1\}^{\leq l}$ to $\{0,1\}^k$.*

Proof: Let $i = (Q, \{\alpha_{j,b}\}_{j\in\{1,\ldots,l\},b\in\{0,1\}}, x_\epsilon)$ be a random index. Since -1 is not a square in $\boldsymbol{Z}^*_P$, the map $\lfloor\cdot\rfloor$ is bijective. Since Q is a $(k+\delta(k))$-bit prime, for a uniformly random value $x \in \boldsymbol{Z}_Q$, the value $x \bmod 2^k$ is statistically close to uniform in $\{0,1\}^k$. It is therefore enough to prove that the output of f_i for random i cannot be distinguished from uniformly random values from Q_P. For this purpose define for $j \in \{1, \ldots, l\}$ and $b \in \{0,1\}$ a function $f_{j,b} : Q_P \to Q_P$, $f_{j,b}(x) = x^{\alpha_{j,b}} \bmod P$ and a function $g_{j,b}$ which is uniformly random from Q_P to Q_P. Then for $m \in \{0, \ldots, l\}$ let $h^m_{j,b} = f_{j,b}$ if $j \geq m$ and let $h^m_{j,b} = g_{j,b}$ otherwise. Finally let $h^m_i(\sigma) = h^m_{l,\sigma_l} \circ \cdots \circ h^m_{1,\sigma_1}(x_\epsilon)$. Then $f_i = h^0_i$ and h^l_i is statistically close to a uniformly random function from $\{0,1\}^{\leq l}$ to Q_P. Only statistically close as collisions will distinguish h^l_i from a uniformly random function: If $|\sigma_1| = |\sigma_2|$ and $h^l_i(\sigma_1) = h^l_i(\sigma_2)$, then for all suffixes σ, $h^l_i(\sigma_1\|\sigma) = h^l_i(\sigma_2\|\sigma)$.

It is therefore enough to prove that h^0_i and h^l_i cannot be distinguished, which can be done by a hybrids argument. Assume namely that there exists $m \in \{1, \ldots, l\}$ such that the functions h^{m-1}_i and h^m_i can be distinguished by a PPT distinguisher D having black-box access to the functions. We will show that this contradicts the DDH assumption. For this purpose, assume that we have access to a black-box o which returns random values of the form $(x, f_{j,0}(x), f_{j,1}(x))$ if $b = 0$ and returns random values of the form $(x, g_{j,0}(x), g_{j,1}(x))$ if $b = 1$. By a simple application of the DDH assumption it can be seen that no PPT algorithm can guess b with anything but negligible advantage. We reach our contradiction by using D to guess b. To be able to do this we show how to generate values $\{x_\sigma\}_{\sigma \in \{0,1\}^{\leq l}}$ distributed as those defined by h_i^{m-1+b} given oracle access to o: Pick all the values x_σ for $\sigma \in \{0,1\}^{\leq m-1}$ as uniformly random values with the only restriction that they are consistent with random functions, i.e. if $|\sigma_1| = |\sigma_2|$ and $x_{\sigma_1} = x_{\sigma_2}$, then for all suffixes σ make sure $x_{\sigma_1\|\sigma} = x_{\sigma_2\|\sigma}$. To generate a value x_σ where $|\sigma| = m-1$, query o and receive a random evaluation (x, x_1, x_2), where x is uniformly random from Q_P. Then let $x_\sigma = x$, let $x_{\sigma\|0} = x_1$, and let $x_{\sigma\|1} = x_2$. Then generate the remaining values x_σ where $|\sigma| > m$ as done in h_i^{m-1} and h_i^m using random exponents. It is straightforward to verify that the values thus defined are distributed as in h_i^{m-1} if $b = 0$ and as in h_i^m if $b = 1$.

To use D to distinguish, run it, and when it queries on $\sigma \in \{0,1\}^{\leq l}$ return x_σ. To make the process efficient, the values x_σ are generated when needed. ■

4 The Multiparty Computation Model

We will study our protocol problems in the framework for universally composable asynchronous multiparty computation from [Can01]. Below we sketch the model.

First the real-life execution of the protocol is defined. Here the protocol π is modeled by n interactive Turing machines (ITMs) $P_1, \ldots, P_n$ called the parties of the protocols. Also present in the execution is an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ modeling the environment in which $\mathcal{A}$ is attacking the protocol. The environment gives inputs to honest parties, receives outputs from honest parties, and can communication with $\mathcal{A}$ at arbitrary points in the execution. Both $\mathcal{A}$ and $\mathcal{Z}$ are PPT ITMs.

Second an ideal process is defined. In the ideal process an ideal functionality $\mathcal{F}$ is present to which all the parties have a secure communication channel. The ideal functionality is an ITM defining the desired input-output behavior of the protocol. Also present is an ideal adversary $\mathcal{S}$, the environment $\mathcal{Z}$, and n so-called dummy parties $\tilde{P}_1, \ldots, \tilde{P}_n$ — all PPT ITMs. The only job of the dummy parties is to take inputs from the environment and send them to the ideal functionality and take messages from the ideal functionality and output them to the environment.

The security of the protocol is then defined by requiring that the protocol emulates the ideal process. We say that the protocol securely realizes the ideal functionality.

The framework also defines the hybrid models, where the execution proceeds as in the real-life execution, but where the parties in addition have access to an ideal functionality. An important property of the framework is that an ideal functionality in a hybrid model can securely be replaced by a sub-protocol securely realizing that ideal functionality.

Below we add a few more details. For a more elaborate treatment of the general framework, see [Can01].

The environment $\mathcal{Z}$ is the driver of the execution. It can either provide a honest party, P_i or $\tilde{P}_i$, with an input or send a message to the adversary. If a party is given an input, that party is then activated. The party can then, in the real-life execution, send a message to another party or give an output to the environment. In the ideal process an activated party just copies its input to the ideal functionality and the ideal functionality is then activated, sending messages to the parties and the adversary according to its program. After the party and/or the ideal functionality stops, the environment is activated again.

If the adversary, $\mathcal{A}$ or $\mathcal{S}$, is activated it can do several things. It can corrupt a honest party, send a message on behalf of a corrupted party, deliver any message sent from one party to another, or communicate with the environment. After a corruption the adversary sends and receives messages on behalf of the corrupted party. We assume a static adversary which corrupts t parties before the execution of the protocol, and then never corrupts again.

The adversary controls the scheduling of the message delivery. In the real-life execution the adversary $\mathcal{A}$ can see the contents of all message and may decide which messages should be delivered and when — it can however not change messages or add messages to a channel. In the ideal process the adversary $\mathcal{S}$ cannot see the contents of the messages as the channels are assumed to be secure. It can only see that a message has been sent and can then decide when the message should be delivered. We will assume that the network is non-blocking. This means that though the adversary is allowed to delay a message for an arbitrary number of activations, any message is eventually delivered if the adversary is activated enough times. If the adversary delivers a message to some party, then this party is activated and the environment resumes control when the party stops.

There is one additional way that the adversary can be activated. An ideal functionality has the opportunity of calling the adversary. This means that the ideal functionality sends a value to the adversary, which then computes a value which is passed back to the ideal functionality. The ideal functionality then proceeds with its actions. Typically, this mechanism is used for modeling some adversarially controlled non-determinism of the ideal functionality. When we specify the functionality for Byzantine agreement, we will use this mechanism to allow the adversary to decide on the result if the honest parties disagree.

At the beginning of the protocol all entities are given as input the security parameter k and random bits. Furthermore the environment is given an auxiliary input z. The environment is then activated and the execution proceeds as described above. At some point the environment stops activating parties and outputs some bit. This bit is taken to be the output of the execution. We use $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ to denote the output of the environment in the real-life execution and use $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$ to denote the output of the environment in the ideal process.

We are now ready to state the definition of securely realizing an ideal functionality. For this purpose let $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble $\{\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\boldsymbol{N},z\in\{0,1\}^*}$ and let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the distribution ensemble $\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\boldsymbol{N},z\in\{0,1\}^*}$.

Definition 5. *We say that* π t-securely realizes $\mathcal{F}$ *if for all real-life adversaries* $\mathcal{A}$*, which corrupts at most* t *parties, there exists an ideal-process adversary* $\mathcal{S}$ *such that for all environments* $\mathcal{Z}$ *we have that the distribution ensembles* $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ *and* $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ *are computationally indistinguishable.*

The following theorem (here stated informally) is proven in [Can01].

Theorem 2. *If* π t*-securely realizes* $\mathcal{F}$ *in the hybrid model with ideal functionality* $\mathcal{G}$ *and* ρ t*-securely realizes* $\mathcal{G}$*, then* π*, with the use of* $\mathcal{G}$ *implemented using* ρ*,* t*-securely implements* $\mathcal{F}$ *in the model without ideal functionality* $\mathcal{G}$*.*

4.1 Some Functionalities

Definition 6. *Let* V *be a PPT algorithm outputting an* $(n+1)$*-tuple of values. A* V-preprocessing model *is a model equipped with an ideal functionality, which when activated the first time generates* $(v_0, v_1, \ldots, v_n) \overset{R}{\leftarrow} V$*, outputs* v_i *to party* P_i*, and outputs* v_0 *to the adversary.*

We will use the preprocessing model for distributing keys for various cryptographic primitives prior to running the actual protocols.

Definition 7. *A* (c,t)-threshold protocol π *for function family* F *is a* t*-secure realization of the functionality* $\mathcal{F}_{F,c}$ *described below.*

Init *On the first activation, the functionality generates* $f \overset{R}{\leftarrow} F$ *and outputs* $(\textit{\textbf{init}})$ *to all parties.*

Evaluate *If a party* P_i *inputs* (j,x)*, we say that the permission to evaluate on* x *is given to* P_j *by* P_i*. The message* (i,j,x) *is output to the adversary and* P_j*. If at some point a total of* c *parties have given permission to some party to evaluate on* x*, then* $(x, f(x))$ *is given to the adversary. If at some point a total of* c *parties have given* P_j *permission to evaluate on* x*, then* $(x, f(x))$ *is given to* P_j*.*

If F *is pseudorandom from* X *to* Y *we call* π *a* (c,t)*-threshold pseudorandom function from* X *to* Y*.*

Definition 8. *The* c-threshold random oracle from X to Y *is the ideal functionality* $\mathcal{F}_{\textit{\textbf{ro}},c,X,Y} = \mathcal{F}_{[X \to Y],c}$ *for evaluating a random function from* X *to* Y*.*

The following theorem is an easy exercise in using the definitions.

Theorem 3. *A* (c,t)*-threshold pseudorandom function from* X *to* Y t*-securely realizes the* c*-threshold random oracle from* X *to* Y*.*

Definition 9. *Let* F *be a family of trapdoor permutations. A* (c,t)-threshold protocol for F *is a protocol* t*-securely implementing the following functionality* $\mathcal{F}_{F,c}$*:*

Init On the first activation, the functionality generates $(f, f^{-1}) \stackrel{R}{\leftarrow} F$ and outputs f to all parties and the adversary.

Invert If a party P_i inputs $(\texttt{invert}, j, x)$, we say that the permission to invert on x is given to P_j by P_i. The message $(\texttt{invert}, i, j, x)$ is output to the adversary and P_j. If at some point a total of c parties have given permission to some party to invert on x, then $(\texttt{invert}, f^{-1}(x))$ is given to the adversary. If at some point a total of c parties have given P_j permission to invert on x, then $(\texttt{invert}, f^{-1}(x))$ is given to P_j.

Definition 10. *The ideal functionality for threshold signatures $\mathcal{F}_{tsig,c}$ is given by the following description.*

Init Let $\mathcal{M}$ be the message space. The functionality keeps for each $M \in \mathcal{M}$ a set $\text{see}(M) \subset \{0, 1, \ldots, n\}$. The interpretation of $\text{see}(M) = P$ is that the parties indexed by P (the adversary is index by 0) see a signature on M. Initially $\text{see}(M) = \emptyset$ for all $M \in \mathcal{M}$.

Sign If a party P_i inputs $(\texttt{sign}, j, M)$, we say that the permission to sign M is given by P_i to P_j. The message $(\texttt{sign}, i, j, M)$ is output to the adversary and P_j. If at some point a total of c parties have given some party permission to sign M, then set $\text{see}(M) = \text{see}(M) \cup \{0\}$ and output $(\texttt{signature}, M)$ to the adversary. If at some point a total of c parties have given P_j permission to sign M, then set $\text{see}(M) = \text{see}(M) \cup \{j\}$ and output $(\texttt{signature}, M)$ to P_j.

Send If a party P_i or the adversary (P_0) inputs $(\texttt{send}, j, M)$ and $i \in \text{see}(M)$, then set $\text{see}(M) = \text{see}(M) \cup \{j\}$ and output $(\texttt{send}, i, M)$ to P_j and the adversary.

Definition 11. *The ideal functionality for Byzantine agreement $\mathcal{F}_{\texttt{ba},t}$ is given by the following description.*

Vote If a party inputs $(\texttt{vote}, vid, b)$, where $b \in \{0, 1\}$, then $(\texttt{vote}, vid, i, b)$ is output to the adversary and we say that the party has voted b in voting vid. The adversary is also allowed to vote.

Decide

The result of voting vid is computed using one of the following rules:

- If $n - t$ parties have voted and $t + 1$ of them voted b and the adversary voted b, then the result is b.
- If $n - t$ honest parties have voted b, then the result is b.
- If $n - t$ honest parties have voted, but do not agree, then the adversary is called to decide the result.

When the result of voting vid has been decided to be b, then $(\texttt{decide}, vid, b)$ is output to all parties and the adversary.

Note, that the three rules for decision are consistent. Especially, if $n-t$ honest parties vote b, then no $t + 1$ parties voted $1 - b$ and therefore the functionality always terminates with decision b.

5 The Threshold DDH-Tree

We now describe our threshold protocol $\pi_{c,\texttt{DDH-Tree}}$ for the DDH-Tree function family.

Key Generation The protocol runs in the preprocessing model for the following values.

- For $i = 1, \ldots, n$, a random public key pk_i for a trapdoor commitment scheme.
- $P = 2Q + 1$, where P and Q are random primes and Q is of length $k + \delta(k)$ bits, where $\log(k)/\delta(k) \in o(1)$.
- g, a random generator of Q_P.
- For $j = 1, \ldots, l$ and $b = 0, 1$:
  - $\alpha_{j,b} \in \boldsymbol{Z}_Q^*$, a uniformly random element.
  - $y_{j,b} = g^{\alpha_{j,b}} \bmod P$.
  - $f_{j,b}(X) \in \boldsymbol{Z}_Q[X]$, a uniformly random degree $c - 1$ polynomial for which $f_{j,b}(0) = \alpha_{j,b}$.
  - For $i = 1, \ldots, n$:
    - $\alpha_{j,b,i} = f_{j,b}(i)$.
    - $y_{j,b,i} = g^{\alpha_{j,b,i}} \bmod P$.
- $x_\epsilon \in Q_P$, a uniformly random element.

The values $(\{pk_i\}_{i=1}^n, Q, g, x_\epsilon, \{y_{j,b}\}_{j=1,b=0}^{l,1}, \{y_{j,b,i}\}_{j=1,b=0,i=1}^{l,1,n})$ are output to all parties and the adversary, and the values $\{\alpha_{j,b,i}\}_{j=1,b=0}^{l,1}$ is output to P_i only.

Evaluation On input $(\texttt{evaluate}, \sigma)$, where $\sigma \in \{0,1\}^l$, the party P_i picks the largest possible prefix σ' of σ for which $x_{\sigma'}$ is known. Then for $j = |\sigma'| + 1, \ldots, l$ the party does the following: The party computes the evaluation share $x_{\sigma[1..j],i} = x_{\sigma[1..(j-1)]}^{\alpha_{j,b,i}} \bmod P$ and sends the value to all parties and proves in zero-knowledge (ZK) to all parties that $\log_{x_{\sigma[1..(j-1)]}}(x_{\sigma[1..j],i}) = \log_g(y_{j,b,i})$ (see below for a description of how to do the ZK-proof). When a party has received evaluation shares and accepted ZK-proofs from all $i \in I$, where $|I| = c$, the party computes $x_{\sigma[1..j]} \leftarrow \prod_{i \in I} x_{\sigma[1..j],i}^{\lambda_{i,I}} = x_{\sigma[1..(j-1)]}^{\alpha_{j,b}} \bmod P$, where the $\lambda_{i,I}$ are the appropriate Lagrange coefficients. When x_σ becomes known, output $\lfloor x_\sigma \rfloor \bmod 2^k$.

ZK-Proofs Assume that P_i knows $\alpha \in \boldsymbol{Z}_Q$ and has sent $g, h, A = g^\alpha, B = h^\alpha$ to P_j, where g and h are generators of Q_P, and wants to prove in ZK that $\log_g(A) = \log_h(B)$. This is done as follows.

Commit Message P_i computes $a \leftarrow g^\beta \bmod P$ and $b \leftarrow h^\beta \bmod P$ for uniformly random $\beta \in \boldsymbol{Z}_Q$, and $c \leftarrow \texttt{commit}_{pk_i}((a,b), r_c)$ for appropriate random bits r_c for the commitment scheme, and sends $(\texttt{commit}, c)$ to P_j.

Challenge P_j generates $e \stackrel{R}{\leftarrow} \boldsymbol{Z}_Q$ and sends e to P_i.

Response P_i computes $z \leftarrow \alpha e + \beta \bmod Q$ and sends (a, b, r_c, z) to P_j.

Check P_j checks that $c = \texttt{commit}_{pk_i}((a,b), r_c)$, $A^e a = g^z \bmod P$, and $B^e b = h^z \bmod P$ and if so, accepts the proof.

Theorem 4. *For (c,t) where $0 \leq t < c \leq n-t$, the protocol $\pi_{c,\texttt{DDH-Tree}}$ is a (c,t)-threshold pseudorandom function from $\{0,1\}^l$ to $\{0,1\}^k$.*

Proof: In [NPR99] Naor, Pinkas and Reingold described a threshold protocol for the *weak* pseudorandom function $x \mapsto x^\alpha \bmod P$ and analysed it for the case $c = t+1$. Subsequently in [CKS00] Cachin, Kursawe and Shoup made a generalization of the proof to handle parameters where $0 \leq t < c \leq n-t$. Our protocol for computing $x_{\sigma[1..(j-1)]}^{\alpha_{j,b}}$ from $x_{\sigma[1..(j-1)]}$ is exactly this protocol, except that we use interactive zero-knowledge proofs, and as detailed below the techniques used in [NPR99,CKS00] generalize straightforwardly to prove our protocol secure. Note, that the theorem specifies input domain $\{0,1\}^l$, and not $\{0,1\}^{\leq l}$. This is to make it secure to reveal $f_i(\sigma')$ for all prefixes σ' of σ when we compute $f_i(\sigma)$. The only difference between our way of computing x_σ from $x_{\sigma'}$ and the protocol used in [NPR99,CKS00] is that we use interactive ZK-proofs. Here we describe how to generalize the analysis to handle this. We show how to simulate the ZK-protocol to an adversary $\mathcal{A}$ while running as an ideal adversary in the ideal process. Say that P_i is giving a proof to P_j. There are two cases:

Assume first that P_i is honest and we as an ideal adversary must give a proof for (g, A, h, B) that $\log_h(B) = \log_g(A) \bmod P$. The simulators used in [NPR99,CKS00] is such that a honest P_i never has to give such a proof without it actually being true. However, the witness $\alpha = \log_g(A) = \log_h(B) \bmod P$ is not always known, so the simulator cannot just run the protocol. We deal with this as follows: As the commit message c we send a trapdoor commitment, which can be opened arbitrarily. When we receive e we then pick $z \overset{R}{\leftarrow} \boldsymbol{Z}_Q$ and compute $a \leftarrow g^z A^{-e} \bmod P$ and $b \leftarrow h^z B^{-e} \bmod P$. Using the trapdoor of the commitment scheme, we then construct random bits r_c such that $c = \texttt{commit}_{pk_i}((a,b), r_c)$ and send (a, b, r_c, z) to P_j. This conversation is distributed exactly as in the protocol.

Assume then that P_i is corrupted. We can then simply run the protocol, as the code of all other parties than P_i is trivial. All that we have to check is that when we accept (g, A, h, B), then indeed $\log_h(B) = \log_g(A) \bmod P$. That this is the case, except with exponentially small probability, is a well-known result[CP92]. ■

The theorem specifies input domain $\{0,1\}^l$. If however the oracle is evaluated on consecutive values $\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$, or more generally, if it is never evaluated on a prefix of a previous input, then the input domain $\{0,1\}^{\leq l}$ would be secure. In that case the loop in **Evaluation** should just be repeated for $j = |\sigma'| + 1, \ldots, |\sigma|$. For consecutive values the worst-case round complexity would be 3 and the worst-case communication complexity would be about $3n^2k$ bits.

If the oracle is evaluated on arbitrary values, then an extra overhead of a factor l will be added. If no bound on the length of inputs is known, or to keep l as low as possible, we can use a family of collision resistant hash-functions: If $F : \{0,1\}^l \to \{0,1\}^m$ is a pseudorandom function and $H : \{0,1\}^* \to \{0,1\}^l$ is a collision resistant hash-function, then $F \circ H : \{0,1\}^* \to \{0,1\}^m$ is again a pseudorandom function, which we can distribute by first hashing the input value locally and then running the threshold pseudorandom function on the hash-value.

In practice a hash-function with output length at least 160 bits would probably by recommended, and so the round complexity would be 480.

Since the time for one round of network communication probably dominates the time to access even a large database, some of this overhead can by removed by preprocessing: Consider the $2m$ key values $\{\alpha_{i,b}\}_{i=j,\ldots,j+m-1,b=0,1}$. If instead of sharing these values we share the 2^m values $\{\alpha_{j,\sigma} = \prod_{i=1}^{m} \alpha_{i+j-1,\sigma_i} \bmod Q\}_{\sigma\in\{0,1\}^m}$, then the computation of $x^{\prod_{j=1}^{l} \alpha_{j,\sigma_i}} \bmod P$ could be speed up by a factor m. By setting $m = 20$ the round complexity could be brought down to 27. The price is a key of about $1Gb$ for $k = 1024$.

6 An RSA and DDH Based Threshold Signature Scheme

In this section we construct a threshold signature protocol $\pi_{\mathtt{tsig},c}$ secure relative to the DDH and RSA assumptions. We will describe the protocol assuming access to a random oracle and an oracle for inverting the RSA function. Using the modular composition theorem, the random oracle can be replaced with our threshold pseudorandom function protocol and the oracle for RSA can be replaced with the protocol given by the following theorem.

Theorem 5. *For the RSA function family with a modulus that is a product of two strong primes and for (c, t), where $n > 3t$ and $t < c \leq n - t$, there exists a statically secure (under the RSA assumption) (c, t)-threshold protocol running in the preprocessing model.*

Proof: In [Sho00] exactly such a protocol is described, which is secure in the model that we consider here, i.e. asynchronous communication and a static adversary. The protocol uses the random oracle model to get non-interactive proofs of equality of discrete logarithms, but we can dispense of the random oracle by doing interactive zero-knowledge proofs. ■

The round complexity of the protocol from [Sho00], using interactive zero-knowledge proofs, is 3, and the communication complexity is about $3n^2k$ bits.

Our threshold signature protocol $\pi_{\mathtt{tsig},c}$ is given by the following description.

Oracles The protocol runs in the hybrid-model with access to an (n, c)-threshold trapdoor permutation functionality $\mathcal{F}_{F,c}$ and a c-threshold random oracle from $\mathcal{M}$ to $\{0, 1\}^k$.

We denote the value output by the random oracle on input M by $H(M)$.

Sign

1. If a party P_i gets input $(\mathtt{sign}, j, M)$, then P_i inputs $(\mathtt{evaluate}, i, M)$ to the random oracle and sends a message to all other parties instructing them to do the same.
2. If the party later sees the output $(\mathtt{evaluate}, M, H(M))$ from the random oracle, then the party inputs $(\mathtt{invert}, j, H(M))$ to $\mathcal{F}_{F,c}$.
3. If a party have received $(\mathtt{evaluate}, M, H(M))$ from the random oracle and $(\mathtt{invert}, f^{-1}(H(M)))$ from $\mathcal{F}_{F,c}$, then the party outputs $(\mathtt{signature}, M)$.

Send

1. If a party P_i gets input $(\texttt{send}, j, M)$ and receives $(\texttt{evaluate}, M, H(M))$ from the random oracle and receives $(\texttt{invert}, f^{-1}(H(M)))$ from $\mathcal{F}_{F,c}$, then P_i sends the message $(\texttt{send}, M, f^{-1}(H(M)))$ to P_j.
2. If a party P_j receives $(\texttt{send}, M, f^{-1}(H(M)))$ from party P_i and receives $(\texttt{evaluate}, M, H(M))$ from the random oracle, then P_j outputs $(\texttt{send}, i, M)$.

Theorem 6. *For $c < n - t$, the protocol $\pi_{tsig,c}$ t-securely realizes the functionality $\mathcal{F}_{tsig,c}$.*

Proof (sketch): We give a slightly informal proof of the theorem, by arguing correctness (if more than c honest parties give a honest P_j permission to sign M, then P_j will eventually output $(\texttt{signature}, M)$) and non-forgeability (if at most c honest parties give any other party permission to sign M, then no honest party will ever output $(\texttt{signature}, M)$ or $(\texttt{send}, i, M)$). Constructing a formal proof by formulating the proof as a simulator is an easy task and is left to the reader.

Correctness: If more than c honest parties give P_j permission to sign M, then because of the non-blocking assumption all these parties will eventually receive $H(M)$ and give P_j permission to invert on $H(M)$. Because of the non-blocking assumption P_j will eventually receive $H(M)$ and $f^{-1}(H(M))$ and will output $(\texttt{signature}, M)$.

Non-forgeability: If at most c honest parties give any other party permission to sign M, then $\mathcal{F}$ will never output $f^{-1}(H(M))$ and especially no honest party will ever output $(\texttt{signature}, M)$. For a honest party to output $(\texttt{send}, i, M)$ the party then has to receive the value $f^{-1}(H(M))$ from (corrupted) P_i. Since $\mathcal{F}$ did not output $f^{-1}(H(M))$ and $H(M)$ is uniformly random, this requires the adversary to break the one-wayness of the trapdoor permutation family F. ■

To sign a message, one call to the oracle functionality and one call to the threshold permutation functionality is used. Using the implementations of Theorems 4 and 5 instead of the ideal functionalities, the round complexity is 6 and the communication complexity is about $6n^2k$ bits, if consecutive values are signed. The overhead over the protocol in [Sho00] is a factor 6. If arbitrary values are signed, the overhead will be in the order $3l + 3$, where l is the output-length of the hash-function used for hashing the messages.

7 Byzantine Agreement

The protocol $\pi_{\texttt{ba},t}$ is given by the following description.

Oracles The protocol runs in the hybrid-model with access to an $(n-t, t)$-threshold signature functionality and a $(t+1, t)$-threshold signature functionality. By *input* $(\boldsymbol{sign}_{n-t}, M)$ we mean *input* $(\boldsymbol{sign}, j, M)$ *to the* $(n-t, t)$-*threshold signature functionality for all* j and we use *input* $(\boldsymbol{sign}_{t+1}, M)$

similarly. The protocol also has access to a $(n-t,t)$-threshold random oracle. By *flip the coin* C_r we mean *input* $(\mathtt{evaluate}, r)$ *to the random oracle and take* C_r *to be the first bit of the output.*

Decide If a party receives a signature on $(\mathtt{main}, r, o)$ for some $r \in \boldsymbol{N}$ and some $o \in \{0,1\}$, then it sends it to all parties and terminates with output $(\mathtt{decide}, vid, o)$. In the remaining description we drop the vote id vid.

Initial Round. On input $(\mathtt{vote}, b_i)$, party P_i follows the code described below.
Initialize the round counter $r \leftarrow 0$.
Input $(\mathtt{sign}_{t+1}, (\mathtt{pre}, 0, b_i))$ and wait for $2t+1$ parties to do the same, i.e. wait until the $(t+1,t)$-threshold signature functionality has output $(\mathtt{sign}, j, (\mathtt{pre}, 0, b_j))$ for $b_j \in \{0,1\}$ for $2t+1$ different j.
Wait for a signature on some $(\mathtt{pre}, 0, b)$ and input $(\mathtt{sign}_{n-t}, (\mathtt{main}, 0, b))$. Wait for $n-t$ parties to do the same.

Flip coin. Let $r = r+1$ and flip coin C_r.

Pre-vote.

1. If for some $b \in \{0,1\}$ a signature on $(\mathtt{pre}, r-1, b)$ is known, then send the signature to all parties and input $(\mathtt{sign}_{n-t}, (\mathtt{pre}, r, b))$.
2. If a signature on $(\mathtt{main}, r-1, \perp)$ is known, then send the signature to all parties and input $(\mathtt{sign}_{n-t}, (\mathtt{pre}, r, C_r))$.

Say that a party P_j has pre-voted when values are received verifying that the party did one of the above.

Pre-collect Wait until $n-t$ parties have pre-voted. This either gives a signature on some $(\mathtt{pre}, r, b)$ (if all pre-voted in the same way or $b = C_r$) or (otherwise) gives a signature on $(\mathtt{pre}, r-1, 1-C_r)$.

Main-vote.

1. If a signature on $(\mathtt{pre}, r, b)$ is known, then send the signature to all parties and input $(\mathtt{sign}_{n-t}, (\mathtt{main}, r, b))$.
2. If a signature on $(\mathtt{pre}, r-1, 1-C_r)$ is known, input $(\mathtt{sign}_{n-t}, (\mathtt{main}, r, \perp))$.

Say that a party P_j has main-voted when values are received verifying that he did one of the above.

Main-collect Wait until $n-t$ parties have main-voted. This either gives a signature on $(\mathtt{main}, r, \perp)$ or gives a signature on some $(\mathtt{pre}, r, b)$.
Go to **Flip coin**.

Theorem 7. *For* t *where* $n > 3t$ *the protocol* $\pi_{\mathtt{ba},t}$ t*-securely realizes* $\mathcal{F}_{\mathtt{ba},t}$ *in the preprocessing model.*

Proof (sketch): We give an informal proof that the ideal functionality and the protocol behaves consistently. A full proof can be constructed along the lines of the proof in [CKS00].

If at least $n-t$ honest parties agree on the value b, then at most t parties will allow to sign $(\mathtt{pre}, 0, 1-b)$ and thus it will not be signed. Therefore all honest parties that come through the initial round have seen $n-t$ parties send $(\mathtt{pre}, 0, b)$ and input $(\mathtt{sign}_{n-t}, (\mathtt{main}, 0, b))$, and will thus terminate with decision b. The non-blocking assumption guarantees that all honest parties terminate. This justifies the second decision rule of the ideal BA functionality. The same line of reasoning shows that if less than $t+1$ parties voted $1-b$, then all parties that terminate, do so after the initial round with output b. However, if less than $n-t$ *honest* parties participates, termination is not guaranteed. This justifies the first decision rule.

To justify the third decision rule, we must argue that if at least $n-t$ honest parties participate, then all honest parties terminate, and agree. For termination, call a round randomizing if when party number $n-t$ contributed to flipping C_r, for one of the values $b \in \{0,1\}$ it was impossible for $(\mathtt{pre}, r-1, b)$ to be signed. We need the fact that out of two consecutive rounds, at least one is randomizing. To see this assume that exactly $n-t$ parties have contributed to C_r and round r is not randomizing. If any of the contributors had seen a signature on $(\mathtt{pre}, r-1, b)$, then $n-t$ parties allowed to sign it and thus at least $n-2t$ honest parties allowed to sign it. This means that at most $2t$ parties would allow to signed $(\mathtt{pre}, r-1, 1-b)$, which will thus never be signed. So, since round r is not randomizing, no party saw a signature on any $(\mathtt{pre}, r-1, b)$. So, the $n-2t$ honest of them are going to input $(\mathtt{sign}_{n-t}, (\mathtt{pre}, r, C_r))$, and thus $(\mathtt{pre}, r, 1-C_r))$ will never be signed, which proves that round $r-1$ is randomizing. Assume then that $n-t$ honest parties have received some input. Since at no point more than $n-t$ parties are waited for, if none of them terminates, arbitrary many rounds will be executed. This means that arbitrary many randomizing rounds are executed, a contradiction as there is a probability of at least $\frac{1}{2}$ that the protocol terminates after a randomizing round (namely if $C_r = 1-b$, where $(\mathtt{pre}, r-1, b)$ is never signed). We then argue agreement. For any party to terminate there must be a least r such that $n-t$ parties allowed to sign $(\mathtt{main}, r, b)$ for some b. This means that $(\mathtt{main}, r, \perp)$ will never be signed and that $(\mathtt{pre}, r, b)$ must have been signed. This means that any honest party that goes through **Main-collect** sees $n-t$ parties go through case one in **Main-vote** and receives a signature on $(\mathtt{main}, r, b)$. The parties that do not get through **Main-collect** will receive a copy of $(\mathtt{main}, r, b)$ sent be a terminating party.

Finally, there is an implicit fourth decision rule: If less that $n-t-t'$ honest parties, where t' is the number of corrupted parties, participate, none of the other rules apply no matter what the adversary does, and thus the ideal functionality will dead-lock. However, in the protocol less than $n-t$ parties will participate, so no $(\mathtt{main}, r, b)$ can get signed, so the protocol behaves accordingly. ■

If the honest parties do not terminate after the initial round (which costs the signing of two messages), they terminate after an average of four iterations of the main loop as every second round is randomizing. One iteration costs the signing of two messages and one random oracle call. Therefore, the expected round complexity will be at most 72 and the expected communication complexity will be about $72n^2k$ bits. These complexities are about a factor 5 larger than for the protocol in [CKS00].

8 Acknowledgments

I'm grateful to Ivan Damgård for many stimulating conversations.

References

[BCF00] Ernest F. Brickell, Giovanni Di Crescenzo, and Yair Frankel. Sharing block ciphers. In Ed Dawson, Andrew Clark, and Colin Boyd, editors, *Information Security and Privacy, 5th Australasian Conference, ACISP 2000, Brisbane, Australia, July 10-12, 2000, Proceedings*, pages 457–470. Springer, 2000.

[BDJR97] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science* [IEE97].

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science*. IEEE, 2001.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 209–218, Dallas, TX, USA, 24–26 May 1998.

[CHH00] Ran Canetti, Shai Halevi, and Amir Herzberg. Maintaining authenticated communication in the presence of break-ins. *Journal of Cryptology*, 13(1):61–106, winter 2000.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC 2000)*, pages 123–132. ACM, July 2000.

[CP92] D. Chaum and T. P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology - Crypto '92*, pages 89–105, Berlin, 1992. Springer-Verlag. Lecture Notes in Computer Science Volume 740.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[Gol01] Oded Goldreich. *The Foundations of Cryptography*, volume 1. Cambridge University Press, 2001.

[IEE97] IEEE. *38th Annual Symposium on Foundations of Computer Science*, Miami Beach, FL, 19–22 October 1997.

[MS95] Silvio Micali and Ray Sidney. A simple method for generating and sharing pseudo-random functions, with applications to clipper-like escrow systems. In Don Coppersmith, editor, *Advances in Cryptology - Crypto '95*, pages 185–196, Berlin, 1995. Springer-Verlag. Lecture Notes in Computer Science Volume 963.

[Nie02] Jesper B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: the non-committing encryption case. In *Advances in Cryptology - Crypto '02*, 2002.

[NPR99] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *Advances in Cryptology - EuroCrypt '99*, pages 327–346, Berlin, 1999. Springer-Verlag. Lecture Notes in Computer Science Volume 1592.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions (extended abstract). In *38th Annual Symposium on Foundations of Computer Science* [IEE97], pages 458–467.

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology - EuroCrypt 2000*, pages 207–220, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1807.