

Improved On-Line/Off-Line Threshold Signatures

Emmanuel Bresson Dario Catalano Rosario Gennaro

DCSSI Crypto Lab, France. emmanuel.bresson@polytechnique.org

Università di Catania, Italia. catalano@dmf.unict.it

I.B.M. T.J. Watson Research Center, USA. rosario@us.ibm.com

PKC'07

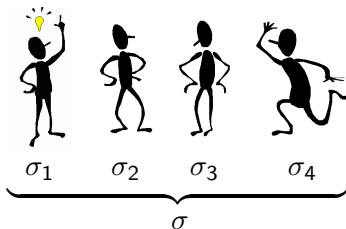
Introduction

- 1 Introduction
 - Motivation
 - Our Result
- 2 Technical issues
- 3 Our Construction
- 4 Proof
- 5 Conclusion

Motivation

Threshold Signatures [DF90]

- signature by n players (rather than by one party)
- hold the secret key in a shared form
- each player produces a *partial signature*
- combine them into σ



Threshold $t < n$: no coalition of t (or less) can sign

Practical applications: large-scale distributed storage [[OceanStore](#), [Pond](#)]

Why (not) Using Threshold Signatures?

The most serious obstacle in the practical deployment of threshold signatures is the time needed to generate signatures

⇒ costs of operations normally required are magnified by the distributed protocol cost

Look for ways of speeding up signature computation, without compromising security

Improving Threshold Signatures

The idea [[CMTW06](#)]: use *on-line/off-line signatures*

A 2-phase signing process:

- 1 **a computationally intensive part**
 - done **off-line** = before the message is known
 - produces some temporary data (stored until the message is known)
- 2 **then computing the signature requires very little effort**
 - **on-line** phase = when getting the message

CMTW06: on-line/off-line threshold signatures

- **Threshold DSS signatures** is an example of such signature: DSS is intrinsically on-line/off-line

What about *generic* solutions: a way to convert any threshold signature into an on-line/off-line one?

Generic constructions



- Proposed by Crutchfield *et al.* [**CMTW06**]
- Combine **any** threshold signature with a threshold version of a specific DLOG-based chameleon hash function
- Reasonably efficient **but** security holds under the *one-more discrete-log assumption*

Our Result

Our Results

A new, improved generic on-line/off-line threshold signature

We also combine any threshold signature (eg, RSA [GJKR00, Sho00]) with a DLOG-chameleon hashing

- Only the standard DLOG assumption (*not 1-more DLOG*)
- Slight increase in the cost of the off-line signature
- Efficiency of the on-line part unchanged

We improve the security assumption without compromising efficiency level

On the Difference Between the Assumptions

p and q two (large) primes s.t. $q|p-1$
 $G = \langle g \rangle$ a subgroup of \mathbb{Z}_p^* with order q

Definition (Discrete logarithm assumption)

Given (p, q, g, g^x) , any PPT \mathcal{A} outputs x with negligible probability

\mathcal{O} a DLOG oracle: $\mathcal{O}(y)$ returns x such that $y = g^x$

Definition (1-more DLOG assumption)

Given p, q and k random values y_1, y_2, \dots, y_k , any PPT \mathcal{A} finds *all* the x_i s.t. $y_i = g^{x_i}$ with negl. probability, provided the oracle is queried at most $k-1$ times.

Newer and not as established as DLOG

Technical issues

- 1 Introduction
- 2 Technical issues
 - Formalization
 - Constructions
 - Difficulties
 - Our Approach
- 3 Our Construction
- 4 Proof
- 5 Conclusion

Formalization

On-line/Off-line Signatures

Definition (Signature scheme)

- *Key generation*: $\text{KeyGen}(1^\ell) = (pk, sk)$
- *Signing*: $\text{Sign}(sk, m) = \sigma$
- *Verifying*: $\text{Ver}(pk, m, \sigma) = 0/1$

Definition (On-line/off-line signature scheme [EGM96])

$\Sigma^{\text{on,off}} = (\text{KeyGen}, \text{Sign}, \text{Ver})$ with Sign divided into 2 phases:

- *Off-line phase*: $\text{Sign}^{\text{off}}(sk) = \sigma^{\text{off}}$
- *On-line phase*: $\text{Sign}^{\text{on}}(sk, m, \sigma^{\text{off}}) = \sigma$

Idea: cost of the on-line phase as small as possible.

On the Other Hand... Threshold Signatures

Definition (Threshold signature scheme)

$T\text{-}\Sigma = (T\text{-KeyGen}, T\text{-Sign}, \text{Ver})$:

- Distributed key generation: $T\text{-KeyGen}(1^\ell) \rightarrow [sk_i](pk)$
- Threshold signing $T\text{-Sign}$ runs in two phases:
 - ① signature share generation $T\text{-Sign}^{\text{share}}[sk_i](m) \rightarrow [\sigma_i]$
 - ② signature reconstruction $T\text{-Sign}^{\text{combine}}[\sigma_i] \rightarrow \sigma$
- verification $\text{Ver}(pk, m, \sigma)$ is unchanged.

Notion independent of the on-line/off-line feature.

Threshold On-line/Off-line Signatures

Definition (On-line/off-line threshold signature)

$T\text{-}\Sigma^{\text{on,off}}$ is made of the following components:

- $T\text{-KeyGen}(1^\ell) \rightarrow [sk_i](pk)$: distributed key generation
- $T\text{-Sign}^{\text{share,off}}[sk_i] \rightarrow [\sigma_i](\sigma^{\text{off}})$: the off-line signature share generation
- $T\text{-Sign}^{\text{combine,on}}[sk_i, \sigma_i](m, \sigma^{\text{off}}) \rightarrow \sigma$: the on-line reconstruction phase
- Ver: verification algorithm (unchanged).

Threshold On-line/Off-line Signatures

- 1 signature share generation coincides with the off-line phase: the obtained “shares” are generated without knowing the message
- 2 signature reconstruction coincides with the on-line phase

Constructions

Construction: Chameleon Hashing

Chameleon hash function [KR00]:

- a public key CH (the actual function)
- a secret trapdoor T
- 2 arguments: message m and random string r

Properties

- 1 collision-resistant, unless one knows the trapdoor T
- 2 knowledge of T allows to find arbitrary collisions
given $c = CH(m, r)$ and arbitrary m' , the holder of the trapdoor can find r' such that $c = CH(m', r')$.

In general, collision-finding is very efficient (a single modular multiplication)

The “Hash-Sign-Switch” Paradigm

Proposed by Shamir-Tauman [ST01] to construct on-line/off-line signatures

Off-line: for random a and r' ,

Hash to obtain $c = CH(a, r')$, then

Sign to obtain σ^{off} , a signature of c

On-line: when given m and with the trapdoor T ,

Switch to m by computing r s.t. $c = CH(m, r)$,

and output (r, σ^{off})

Verif: compute c and verify σ^{off} on it

The Contribution of Crutchfield *et al.*

From the paradigm $c = CH(a, r') = CH(m, r)$ and $\sigma^{\text{off}} = \text{Sign}(c)$:

A way to compute the values c and r distributively

- 1 Pedersen's commitment [Ped91] as chameleon hashing:

$$c = CH(a, r') = g^{r'} h^a$$

- 2 σ^{off} computed on c with a *threshold* signature
- 3 Servers hold shares of $T = \log_g h$ and *distributively* extract r

$$c = g^r h^m$$

$$\implies \sigma = (r, \sigma^{\text{off}})$$

How to “thresholdize” CH

Use techniques from discrete-log based threshold cryptography

Difficulties

Security

Definition (Simulatability)

Guarantees that the threshold signature scheme is as secure as its centralized version [**GJKR98**]

Simulation:

A forger against the threshold scheme is transformed via a simulation of the distributed environment into a forger for the centralized scheme, or a collision-finder for CH

Difficult Issues in their Approach

A technical difficulty prevents a “direct” proof

- 1 Distributiveness: $c = H(m, r)$ is revealed in the off-line phase
- 2 \mathcal{A} learns it *before* the final signature is computed
 - opposed to the centralized Shamir-Tauman solution

Consequences:

- 1 simulation of on-line phase is constrained to a specific c
- 2 this c is generated before m was known.

This is why [**CMTW06**] must use 1-more DLOG assumption

Our Approach

Our Approach

An alternative approach, avoiding the stronger assumption

Basic idea = variation of Pedersen's commitment

$$\text{Define } CH(m, r, s) = g^m h_1^r h_2^s$$

Can be “thresholdized” as well

Crucial property: two “independent” trapdoors ($\log_g h_1$ and $\log_g h_2$)

- give a random one to the simulator to help in . . . simulating
- if \mathcal{A} finds a collision for CH , with probability $1/2$ this will reveal the other trapdoor

A technique going back to the seminal Goldwasser, Micali and Rivest [[GMR88](#)]

Our Approach (cont.)

Compute and sign $c = H(a, r, s) = g^a h_1^r h_2^s$
 $\sigma^{\text{off}} = (c, \text{sign}(c))$

Share the value $\log c = \tau = a + r \cdot y + s \cdot z$ **between players**
 with $y = \log_g h_1$ and $z = \log_g h_2$ shares $[\tau_i]$

Prepare a random shared value s' shares $[s'_i]$

Add ω_i **a** $2t$ -share **of 0 for uniformity** shares $[\omega_i]$

When given m **find a collision:** $c = g^a h_1^r h_2^s = g^m h_1^{r'} h_2^{s'}$
 $\tau = m + y \cdot r' + z \cdot s'$

Distributively compute r'

$$r' = (\tau - m - zs')Y + \omega_i \quad \text{where } Y = y^{-1}$$

$$\sigma = (c, \text{sign}(c), r', s')$$

On the Robustness

[CMTW06] is robust provided that $n > 3t + 1$

Definition (Robustness)

If the protocol is secure even in the presence of t arbitrarily malicious players, then the protocol is called *robust*.

Our scheme is robust against up to:

- 1 $n/3$ players, that can be *halting* at any time
- 2 $n/4$ players, that can be *malicious* at any time
- 3 $n/3$ players, that can be *malicious* except on-line
- 4 $n/3$ players, that can be *malicious* at any time, with a slight loss of efficiency

Our Construction

- 1 Introduction
- 2 Technical issues
- 3 Our Construction
 - Building Blocks
 - Our New Scheme
 - Security
- 4 Proof
- 5 Conclusion

Building Blocks

Computation on shared secrets

- $MUL[a_i, b_i] \rightarrow [c_i]$: well-known protocol [BGW88]
Shares of $c = a \times b$ from those of a and b
- $INV[a_i] \rightarrow [b_i]$: due to [BB88]
Shares of a^{-1} from those of a
 - 1 Players jointly generate a shared random r
 - 2 They multiply a and r
 - 3 They jointly reconstruct $u \equiv ar \pmod{q}$
 - 4 If $u \equiv 0 \pmod{q}$ the protocol is restarted
 - 5 Else each P_i locally computes his share of $b = a^{-1} \pmod{q}$ as $b_i = r_i \cdot u^{-1} \pmod{q}$

These protocols can be made robust easily

Pedersen's Verifiable Secret Sharing

Verifiable, robust version of [Sha79]

$G = \langle g \rangle = \langle g_1 \rangle$ of order q in \mathbb{Z}_p^* and $pub = (g, g_1, p, q, t, n)$

- Ped-VSS(pub) $\langle a, r \rangle \rightarrow [a_i, r_i](V)$ [Ped91]

Share a value a using randomness r

- 1 random $f(\cdot)$ and $g(\cdot)$ of degree t , with $f(0) = a$
- 2 shares are $(a_i, r_i) = (f(i), g(i))$ for each P_i
- 3 verification using $V_j = g^{\alpha_j} g_1^{\beta_j}$ with α_j, β_j coeff. of f and g

- Joint-RPed-VSS(pub) $\rightarrow [a_i, r_i](T_S, V)$

Generalization: no special dealer

- 1 each P_i shares a random secret $a^{(i)}$
- 2 each player adds the shares he received
- 3 so that the final secret is $a = \sum_i a^{(i)}$ (unknown)

Other Distributed Primitives

- $\text{DL-DKG}(pub) \rightarrow [x_i](y, V)$ [GJKR98]
Generate public key $y = g^x$ with shared private key $x = [x_i]$
- $\text{Share-Exp}[a_i, b_i, c_i](g, h_1, h_2) \rightarrow (g^a h_1^b h_2^c)$ [DG03]
Compute $g^a h_1^b h_2^c$ when a, b, c are shared secrets

Our New Scheme

The new scheme

$T\text{-}\Sigma^{\text{off, on}}$ is composed of the following algorithms

$$T\text{-}\Sigma^{\text{off, on}} = (T\text{-KeyGen}, T\text{-Sign}^{\text{share, off}}, T\text{-Sign}^{\text{combine, on}}, \text{Ver})$$

- Based on a threshold signature scheme $(T\text{-KGen}, T\text{-Sig}, \text{Ver})$
- Public parameters $pub = g, g_1, p, q, t, n$
- We assume that $t < n/4$.

Key Generation

Common Output: the public key PK of the scheme

Private Output for P_j : a share SK_j of the signing key

- 1 run T-KGen, obtain a public key vk , each player receives a share sk_i of sk
- 2 run DL-DKG(pub) algorithm twice, obtain h_1, h_2 . Let y_i, z_i the shares of secret keys y, z s.t. $g^y = h_1$ and $g^z = h_2$
- 3 run $INV[y_i] \rightarrow [Y_i]$, obtain shares Y_i of $Y = y^{-1}$
- 4 set $PK = (pub, vk, h_1, h_2)$, while each P_i sets $SK_i = (sk_i, y_i, z_i, Y_i)$ as its local secret key

Off-line Signing

Private Input for P_j : $SK_j = (sk_j, y_j, z_j, Y_j)$

Private Output for P_j : signature token σ^{off} + signature share σ_j

- 1 run Joint-RPed-VSS(pub) 3 times to produce a, r, s , whose shares are a_i, r_i, s_i
- 2 run Share-Exp[a_i, r_i, s_i], each P_i with a_i, r_i, s_i , obtain publicly $\text{Com} = g^a h_1^r h_2^s$
- 3 run T-Sig($\text{Com}, [sk_i]$) to get a signature ρ on Com .
- 4 run (a simplified) Joint-RPed-VSS to generate shares ω_i of 0, through a $2t$ -degree (random) polynomial p_0
- 5 run MUL[r_i, y_i] and MUL[s_i, z_i] to get shares of $r \cdot y$ and $s \cdot z$. Then, **locally**, compute shares τ_i of $a + r \cdot y + s \cdot z$
- 6 run Joint-RPed-VSS(pub) to get shares s'_i of a random s'
- 7 set $\sigma^{\text{off}} = (\text{Com}, \rho)$ and $\sigma_i = (\omega_i, \tau_i, s'_i)$

On-line Signing

Public Input: message m to be signed

Private Input for P_j : the signing key $SK_j = (sk_j, y_j, z_j, Y_j)$, the token $\sigma^{\text{off}} = (\text{Com}, \rho)$ and the share $\sigma_i = (\omega_i, \tau_i, s'_i)$

Public Output: a signature σ for m

- ① broadcast s'_i , thus recover s' .
- ② set $r'_i = (\tau_i - m - s' \cdot z_i) \cdot Y_i + \omega_i \bmod q$
- ③ r'_i , thus reconstructing r' .
- ④ the signature for m is $\sigma = (\text{Com}, \rho, r', s')$

Verification

Given (Com, ρ, r, s) and a message m , one accepts it as valid if:

$$\text{Ver}(vk, \text{Com}, \rho) \stackrel{?}{=} 1 \quad \wedge \quad \text{Com} \stackrel{?}{=} g^m h_1^r h_2^s$$

Security

Security Theorem

Theorem

Assuming that

- 1 $T\text{-}\Sigma = (T\text{-KGen}, T\text{-Sig}, \text{Ver})$ secure against adaptive chosen message attack
- 2 DLOG assumption holds in G
- 3 \mathcal{A} is static and controls up to $t < n/4$ participants.

Then

the On-Line/Off-line Threshold Signature is existentially unforgeable against an adaptive chosen message attack

Proof

- 1 Introduction
- 2 Technical issues
- 3 Our Construction
- 4 Proof
 - Overview
 - Type I Forgeries
 - Type II Forgeries
 - Robustness
- 5 Conclusion

Overview

Proof Main Line

A classical reductionnist proof

- 1 We assume there exists an adversary \mathcal{A} against the existential unforgeability of our scheme
- 2 We show how to break:
 - either the unforgeability of the underlying signature T - Σ
 - or the discrete logarithm assumption

Main Branches of the Proof

Any valid forgery $(\text{Com}, \rho, s', r')$ on a message m' must be of one of the following (mutually exclusive) types

- **Type I:** $\text{Com} \neq \text{Com}_i$ for all previously issued signatures $(\text{Com}_i, \rho_i, s'_i, r'_i)$ on messages m'_i ,
- **Type II:** $\text{Com} = \text{Com}_i$ for some previously issued signature $(\text{Com}_i, \rho_i, s'_i, r'_i)$ on a message m'_i , but at least two of the following hold
 - ① $m' \neq m'_i$
 - ② $s' \neq s'_i$
 - ③ $r' \neq r'_i$

$$(\text{Com} = g^m h_1^r h_2^s)$$

Type I Forgeries

Reduction for Type I Forgeries

Assume \mathcal{A} produces a Type I forgery: $\text{Com} \neq \text{Com}_i, \forall i$
Build a \mathcal{B} against unforgeability of T- Σ

We use the fact that (according to [GJKR98]):

- 1 T-KGen is simulatable: a simulator S_1 that, given the public output of a T-KGen run, simulates \mathcal{A} 's view
- 2 T-Sig is simulatable: a simulator S_2 that, given t shares and the signature σ , simulates \mathcal{A} 's view

Type I: Simulation of T-KeyGen

On-line/Off-line Threshold Key Generation:

- 1 Step 1: \mathcal{B} runs S_1 on input the public parameters pub to simulate generation of sk_i 's
- 2 \mathcal{B} runs steps 2, 3 and 4 honestly, for y_i, z_i and Y_i 's
- 3 Due to T-KGen's simulatability, the entire simulation of T-KeyGen is indistinguishable from real

Type I: Simulation of T-Sign^{share,off}

Off-line Threshold Signing Protocol:

- 1 Steps 1 (share m, r, s) and 2 done honestly: compute $\text{Com} = g^m h_1^r h_2^s$
- 2 Step 3: \mathcal{B} queries his T-Sig-signing oracle to get a signature ρ_i on the computed Com_i
- 3 \mathcal{B} runs simulator S_2 on input the controlled players' shares and ρ_i
- 4 Steps 4 (share 0 through $[\omega_i]$), 5 (compute shares τ_i) and 6 (sharing s') done honestly

Type I: Simulation of T-Sign^{combine,on}

On-line Threshold Signing Protocol:

Whenever \mathcal{A} asks the i -th query on a message m'_i , \mathcal{B} runs the protocol honestly

Type I: Conclusion

- \mathcal{A} produces a forgery of type I $(\rho, \text{Com}, s', r')$
- Type I forgery:
 - ① Com differs from all Com_i ;
 - ② thus never queried by \mathcal{B} to its signing oracle

Then: \mathcal{B} produces its own forgery against $\text{T-}\Sigma$ by setting

$$M = \text{Com} \quad \text{and} \quad \text{Sig} = \rho$$

Type II Forgeries

Reduction for Type II Forgeries

Assume \mathcal{A} produces a Type II forgery
Build a \mathcal{B} computing a discrete log h in a base g

\mathcal{B} knows the discrete log of g_1 to the base g and can simulate DL-DKG

We assume, for simplicity, that $m' \neq m'_i$ (extending the proof to m' (the forged message) may be equal to m'_i (the message queried for signing) is easy)

Preliminary Operation

\mathcal{B} flips a coin β

$\beta = 0$: bet on the fact that \mathcal{A} will provide a forgery of type II where

$$m' \neq m'_i \quad \text{and} \quad r' \neq r'_i$$

$\beta = 1$: bet on the fact that the forgery will satisfy:

$$m' \neq m'_i \quad \text{and} \quad s' \neq s'_i$$

Type II: Simulation of T-KeyGen for $\beta = 0$

- ① Step 1 is done honestly: \mathcal{B} knows everything
- ② Step 2:
 - ① DL-DKG(pub) $\rightarrow [y_i](h_1, V)$ is replaced by the simulator $S(g, h)$ for DL-DKG. This forces $h_1 = h$
 - ② however the players share a secret \hat{y} which is not $y = \log_g h_1$
 - ③ for h_2 , play honestly
- ③ Step 3: run $\text{INV}[\hat{y}_i] \rightarrow [\hat{Y}_i]$ instead of $\text{INV}[y_i] \rightarrow [Y_i]$
- ④ Step 4 as in the real game

Type II: Simulation of T-Sign^{share,off} for $\beta = 0$

- ① Steps 1 (share m, r, s), 2 (compute $\text{Com} = g^m h_1^r h_2^s$), 3 ($\rho = \text{T-Sig}(\text{Com})$) and 4 (a share of 0) are done honestly
- ② Step 5: honestly except that $\text{MUL}[r_i, y_i]$ is run using $[r_i, \hat{y}_i]$
- ③ Step 6 (share s') is done honestly

Type II: Simulation of T-Sign^{combine,on} for $\beta = 0$

- ① Recover s' (possible as $n - t > 3n/4 > t$)
- ② Once m' is known, it set $r' = r$ and compute \hat{s}' such that

$$r' = (m + r\hat{y} + sz - m' - \hat{s}'z)\hat{Y} \bmod q$$

with \hat{Y} computed during key generation

Hence one has:

$$\hat{s}' = (m - m')z^{-1} + s \bmod q$$

\mathcal{B} knows $\log_g g_1$ and can cheat to interpolate s' as \hat{s}'

Type II: conclusion for $\beta = 0$

Adversary produces a type II forgery such that:

- 1 $m'_i \neq m'$
- 2 $r'_i \neq r'$

Then we recover the challenged $\log_g h$:

- 1 $\text{Com}' = \text{Com}_i$
- 2 then

$$g^{m'_i + zs'_i} h_1^{r'_i} = g^{m' + zs'} h_1^{r'}$$

- 3 and thus

$$\log_g h = ((m'_i - m') + z(s'_i - s'))(r' - r'_i)^{-1} \bmod q$$

Type II: simulation for $\beta = 1$

If $\beta = 1$ \mathcal{B} hopes \mathcal{A} provides a type II forgery with:

$$m' \neq m'_i \quad \text{and} \quad s' \neq s'_i$$

Key Generation: simulation to generate h_2 : thus \mathcal{B} knows a value

$$\hat{z} \neq \log_g h_2$$

Note, $Y = y^{-1}$ is honestly computed

Off-line Signing: Just switch the roles of z and y

On-line Signing: \mathcal{B} knows $\log_g g_1$ to interpolate s' as the s shared in round **1**

Robustness

Achieving Robustness

Key generation and off-line signing: robustness w.r.t. up to $\lfloor n/3 \rfloor$ players

Verification $\text{Ver}(vk, \text{Com}, \rho)$ fails: some participants are providing incorrect shares

Assuming $n > 4t$ gives us enough points to correctly interpolate s' and r' and reconstruct the correct signature

Trivial approach: try all possible subsets of $2t + 1$ shares
The number of such subsets is exponential (in n)

Achieving Robustness

We suggest the following two-phases approach.

First Phase: check correctness of the s_i'' 's, via commitments in off-line round **6**

If t incorrect shares, remove all dishonest players immediately

Second Phase (If less than t incorrect shares): in on-line round **3**, players interpolate r' using Berlekamp-Welch decoder [BW]:

$d = 2t$ degree polynomial, up to $f = t$ erroneous points, correct interpolation requires $d + 2f + 1$, which means at least $2t + 2t + 1 = 4t + 1$

Improving Robustness for up to $t < n/3$ Faults

Slightly modify the algorithms to improve robustness

Key Generation: additionally compute shares $\lambda_i = Y_i \cdot z_i$

Off-line Signing: additional run of $\text{MUL}[\tau_i, Y_i] \rightarrow \mu_i$ to create shares of $\tau \cdot Y$

On-line Signing: modify step 2 by setting

$$r'_i = \mu_i - mY_i - s'\lambda_i + \omega_i \text{ mod } q.$$



- Threshold improves to $t < n/3$ malicious players at any time
- The proof remains basically identical
- The modified protocol is less efficient, but the loss involves the off-line components only

Conclusion

- 1 Introduction
- 2 Technical issues
- 3 Our Construction
- 4 Proof
- 5 Conclusion

Conclusion

A new on-line/off-line threshold scheme

- Improved security: standard assumption
- On-line efficiency maintained
- Reasonable lost in off-line phase
- Robustness decreases to $t < n/4$
- ... but can be maintained to $n/3$ at a small cost

References I



J. Bar-Ilan and D. Beaver.

Non-Cryptographic Fault Tolerant Computing in a Constant Number of Rounds of Interaction.

In *PODC '89*, pp.201–209.



M. Ben-or, S. Goldwasser and A. Wigderson

Completeness Theorems for Non-Cryptographic Fault Tolerant Distributed Computation.

In *STOC' 88*, pp.1–10.



E. Berlekamp and L. Welch

Error Correction of Algebraic Block Codes.

US Patent 4,633,470.



C. Crutchfield, D. Molnar, D. Turner and D. Wagner

Generic On-Line/Off-Line Threshold Signatures

In *PKC '06*, pp.58–74.

References II



I. Damgård and K. Dupont.

Efficient Threshold RSA Signatures with General Moduli and No Extra Assumptions.

In *PKC '05*, pp 346–361.



Y. Desmedt and Y. Frankel.

Threshold Cryptosystems.

In *Crypto'89*, pp.307–315.



M. Di Raimondo and R. Gennaro

Provably Secure Threshold Password-Authenticated Key Exchange.

In *Eurocrypt'03*, pp.507–523.



S. Even, O. Goldreich and S. Micali.

On-Line/Off-Line Digital Signatures.

J. Cryptology 9(1):35-67, 1996.



P. Feldman

A Practical Scheme for Non-Interactive Verifiable Secret Sharing.

In *FOCS '87*, pp. 427–437.

References III



R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin.
Secure Distributed Key Generation for Discrete-Log Public-Key
Cryptosystems.
Eurocrypt'99, pp.295–310.



R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin.
Robust and Efficient Sharing of RSA Functions.
J. Cryptology 13(2):273-300, 2000.



R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin.
Robust Threshold DSS Signatures.
Inf. Comput. 164(1): 54-84, 2001.



R. Gennaro, M. Rabin and T. Rabin.
Simplified Vss and Fast-Track Multi-Party Computations With
Applications to Threshold Cryptography.
In *Proc. PODC '98*, pp.101–111.

References IV



S. Goldwasser, S. Micali and R. Rivest.

A Digital Signature Scheme Secure Against Adaptive Chosen Message Attacks.

SIAM J. on Computing 17(2):281-308, 1988



H. Krawczyk and T. Rabin.

Chameleon Signatures.

NDSS 2000, pp.143-154.



National Institute for Standards and Technology.

Digital Signature Standard (DSS).

Technical Report 169, 1991.



J.Kubiatowicz, D.Bindel, Y.Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells and B. Zhao.

OceanStore: An architecture for GlobalScale Persistent Storage.

In *ASPLOS '00*, pp.190-201.

References V



T. Pedersen.

Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing.

Crypto '91, pp.129-140.



S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao and J. Kubiatowicz.

Pond: The OceanStore prototype.

In *FAST '03*.



A. Shamir.

How to share a secret.

CACM, 22(11):612–613.



A. Shamir and Y. Tauman.

Improved On-line/Off-line Signature Schemes.

Crypto '01, pp.355-367.

References VI



V. Shoup.
Practical Threshold Signatures.
Eurocrypt '00, pp.207–220.