Asynchronous Proactive Cryptosystems without Agreement

Stas Jarecki (UC Irvine)
Bartosz Przydatek (ETH Zurich, Switzerland)
Reto Strobl (Google, Switzerland)

Proactive Cryptosystems

Motivation:

weakest link in a public key cryptosystem is often the server that ‘runs’ the cryptosystem

Goal of proactive cryptosystems:

run a ‘conventional’ public key cryptosystem in a more fault-tolerant and secure way

Proactive Cryptosystems

Main idea: distribution + periodic refresh [OY91]

distribution:

- distribute secret key among n servers (setup)
- perform cryptographic operation by multiparty protocol

Goal: small fraction of servers cannot learn the secret key or make the protocol fail

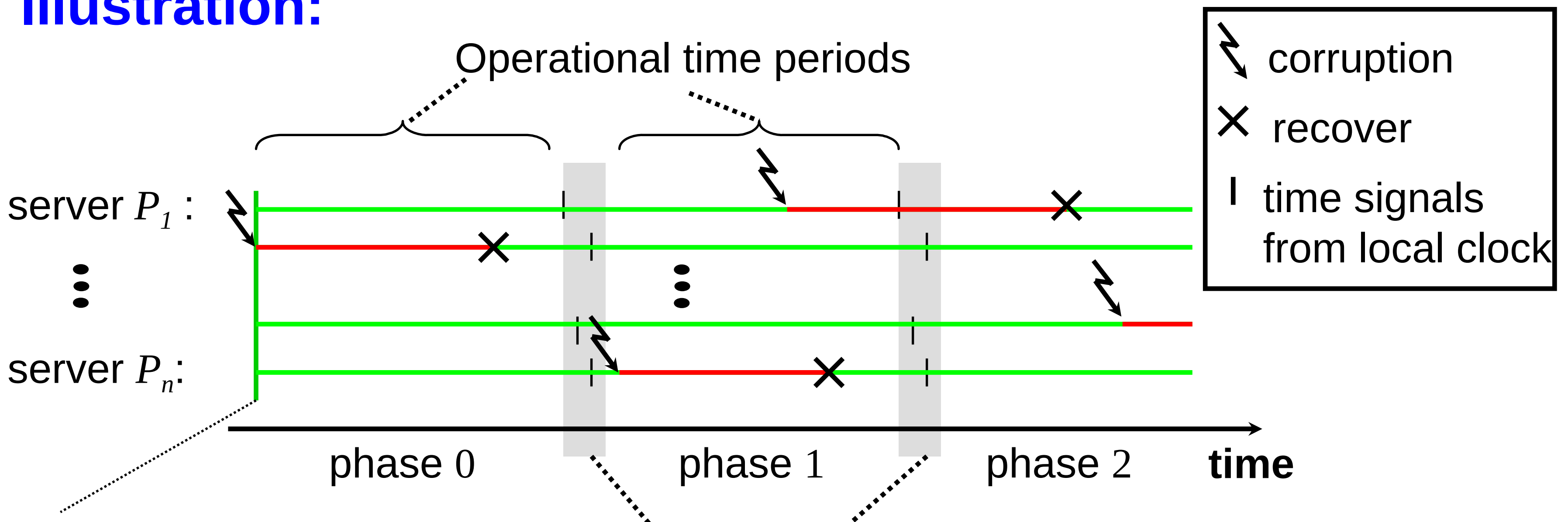
refresh:

periodically refresh shares of secret key

Goal: re-establish security of servers that recovered from a corruption
(recovery may occur by means of external mechanisms)

Asynchronous Proactive Cryptosystems

Illustration:



setup:
key is shared among all servers

periodic refresh:
servers execute protocol to refresh shares
(protocol triggered upon signal of local clock)

Security guarantees:

the ‘proactivized’ cryptosystem is secure if no large fraction of servers is corrupted between two refreshes

Overview of the Paper

Contents:

set of protocols for proactivizing Discrete Logarithm based cryptosystems over asynchronous network

secure if adversary crashes or eavesdrops $t < n/3$ in every two subsequent phases (no Byzantine corruption)

Novelty:

protocols do not rely on Byzantine agreement

→ surprising… (contradicts a folklore believe)

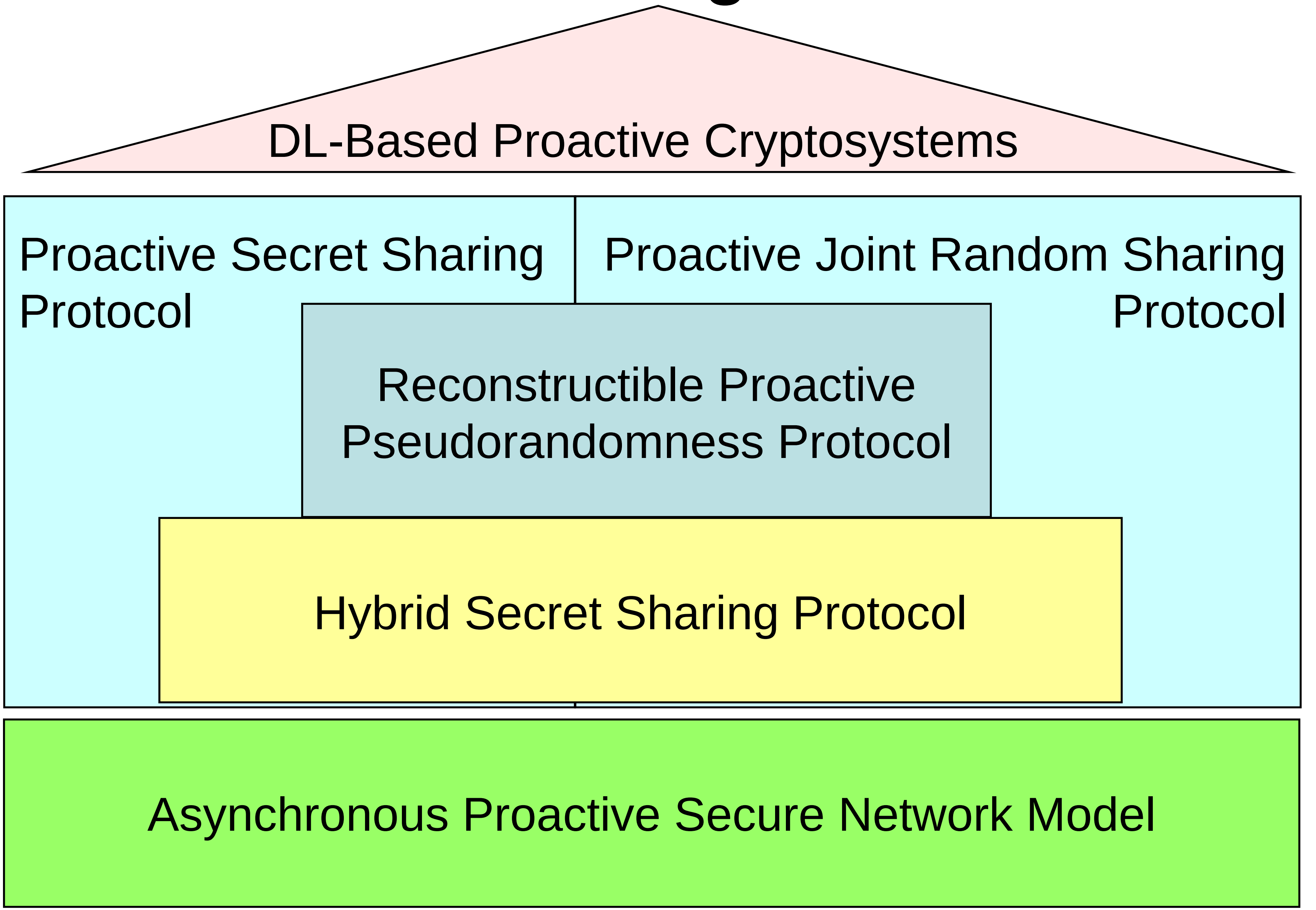
→ bounded ***worst-case*** complexity
(before only bounded average case)

→ worst-case round-complexity = 3 times smaller than average-case complexity of previous soultions

Outline of the talk

- Introduction to proactive cryptosystems
- An overview of the proposed construction
- Protocols
  - Hybrid Secret Sharing
  - Reconstructible Proactive Pseudorandomness
  - Proactive Secret Sharing and Joint Random Secret Sharing
- An example application: Proactive Schnorr's Signatures
- Conclusions

The Building Blocks



Outline of the talk

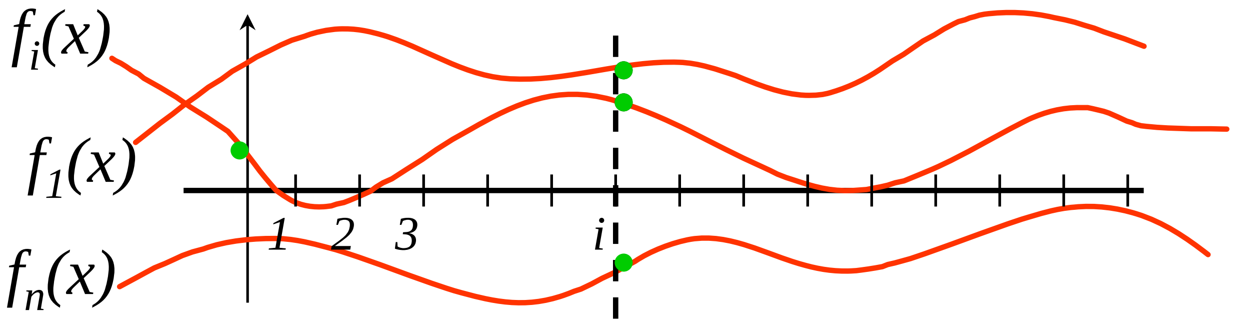
- Introduction to proactive cryptosystems
- An overview of the proposed construction
- Protocols
  - Hybrid Secret Sharing
  - Reconstructible Proactive Pseudorandomness
  - Proactive Secret Sharing and Joint Random Secret Sharing
- An example application: Proactive Schnorr's Signatures
- Conclusions

Hybrid Secret Sharing Protocol

Input: k-bit secret s and k-bit randomness r

Output of server i:

let $f_1(x),\ldots, f_n(x)$ denote pseudo-random t-degree polynomials over $F_{2^k}[x]$ s.t. $f_1(0)+\ldots+ f_n(0) = s$



Server i outputs the green values, i.e., $f_i(0), f_1(i),\ldots, f_n(i)$

Properties:

- servers only learn their input and output
- either all or no server terminates
- protocol is deterministic

Outline of the talk

- Introduction to proactive cryptosystems
- An overview of the proposed construction
- Protocols
  - Hybrid Secret Sharing
  - Reconstructible Proactive Pseudorandomness
  - Proactive Secret Sharing and
    Joint Random Secret Sharing
- An example application: Proactive Schnorr's Signatures
- Conclusions

Reconstructible Proactive Pseudorandomness (RPP) Scheme

Goal:

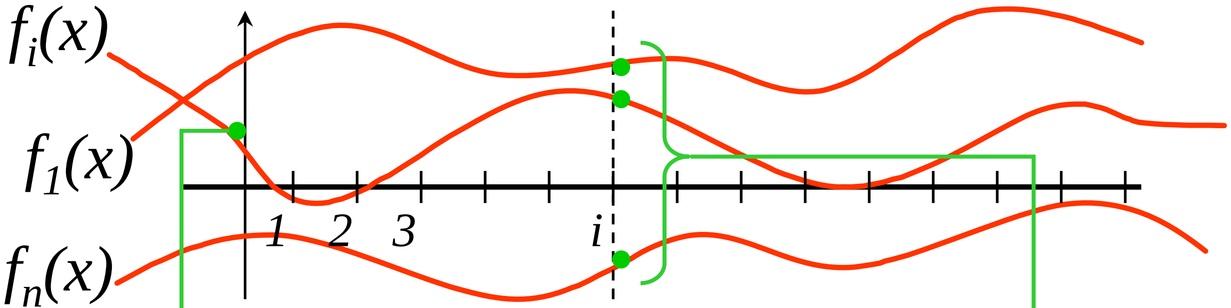
- provide at every phase τ every server P_i with a new, secret pseudo-random value $pr_{\tau,i}$

- allow any set of $(n\text{-}t)$ servers to reconstruct the value $pr_{\tau,j}$ of any server P_j

RPP Scheme Implementation

Setup (by trusted dealer):

1) choose n polynomials of degree n-t at random over $F_{2^k}[x]$



2) give to every server P_i the green values in the picture

k-bit key $r_i=f_i(0)$

$(n$-$t)$-out-of-n backup share $r_{ji}=f_j(i)$ of every other server's key r_j,

RPP Scheme Implementation

Idea:

compute $pr_{\tau,i}$ as $\varphi_{r_i}(c)$ for some constant c, where $\{\varphi_i\}$ is a distributed pseudorandom function family

$\rightarrow$ pseudo-randomness and reconstructability of $pr_{\tau,i}$ follows from the distribution of r_i and properties of $\{\varphi_i\}$

- for a random key r, $\varphi_r(v)$ looks random for any v
- if $r_1,\ldots,r_n$ are polynomial $(n\text{-}t)\text{-}out\text{-}n$ shares of r, then $\varphi_r(v)$ can be computed from any $(n\text{-}t)$-sized subset of $\varphi_{r_1}(v),\ldots,\varphi_{r_n}(v)$
- for efficient such functions, see [Nie02]

Remaining Issue: refresh keys r_i and backup shares!

RPP Scheme Implementation

Refreshing keys and backup shares (steps of server P_i):

1) upon phase change:

share $\varphi_{r_i}(a)$ using randomness $\varphi_{r_i}(b)$, where a,b are public constants, and r_i is current key

2) upon terminating $(n\text{-}t)$ sharing protocols:

reveal $\varphi_{r_{mi}}(a)$, $\varphi_{r_{mi}}(b)$, for dealers m with pending sharings

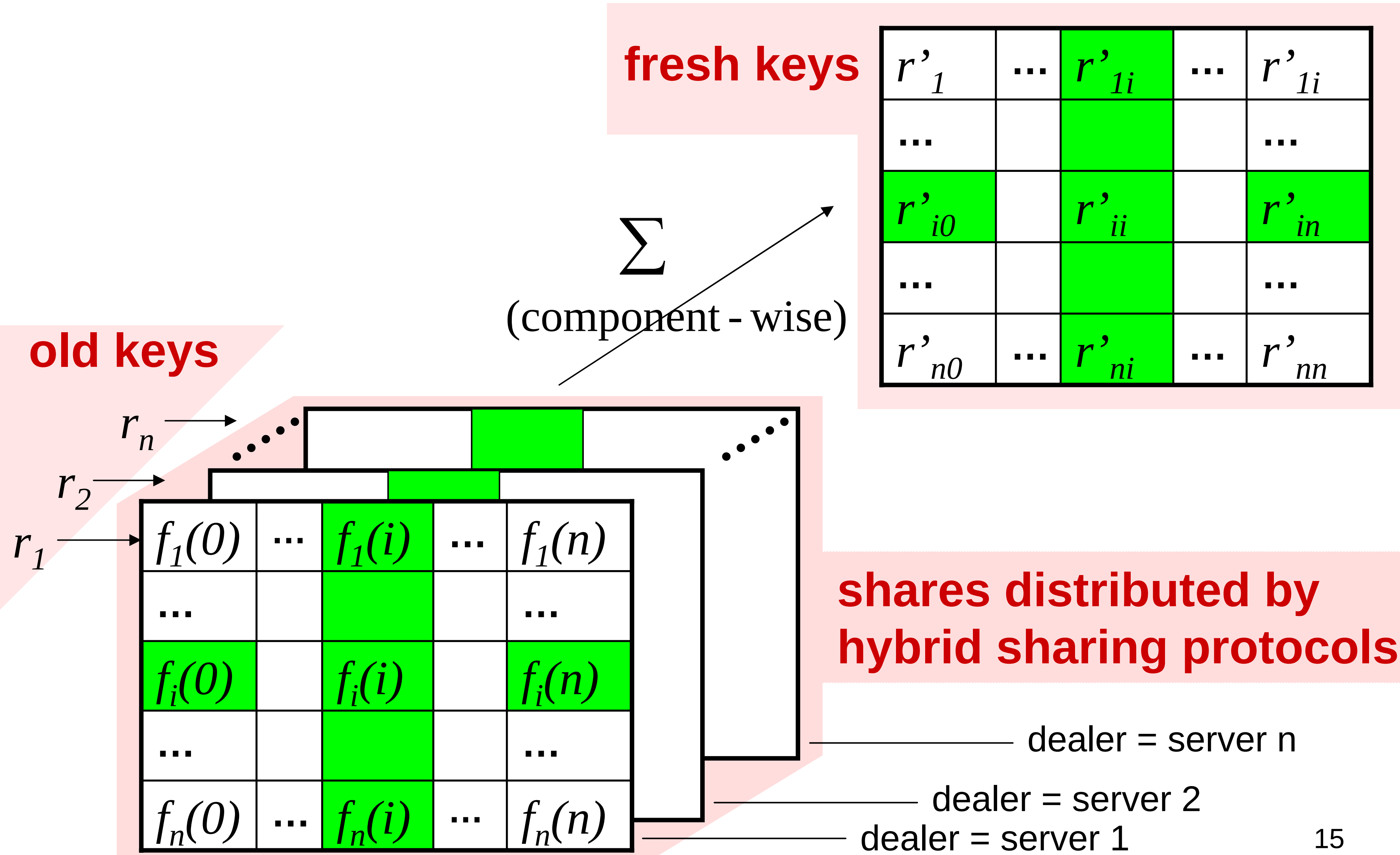
3) upon receiving (n-t) 'shares' $\varphi_{r_{mi}}(a)$ $\varphi_{r_{mi}}(b)$, for some m:

compute $\varphi_{r_i}(a)$ and $\varphi_{r_i}(b)$, and complete sharing locally

4) upon terminating all sharing protocols:

fresh key r'_i= sum of all additive shares
fresh backup share r'_{mi} = sum of all received backup shares for server P_m

Refreshing the keys (illustration)



RPP Properties

Correctness:

pevious picture = situation when all sharing protocol terminate

BUT:

- What if certain sharing protocols do not terminate?

- Don’t servers need to agree on which sharing protocols terminate, and which have to be reconstructed locally?

NO!

→ since sharing is deterministic, protocol and “local reconstruction” yield the same shares! (stil the same picture)

RPP Properties

Pseudo-randomness:

Lots of information gets revealed

Why are fresh keys pseudo-random?

Claim: The old key of at least one honest server remains hidden from the adversary.

Argument:

By eavesdropping, adversary learns t old keys and t backup shares in the remaining $(n-t)$ old keys

To learn all old keys, she needs $n-2t$ backup shares in the $n-t$ remaining old keys

Honest servers reveal only $t\ (n-t) < (n-2t)$ backup shares!

Outline of the talk

- Introduction to proactive cryptosystems
- An overview of the proposed construction
- Protocols
  - Hybrid Secret Sharing
  - Reconstructible Proactive Pseudorandomness
  - Proactive Secret Sharing and Joint Random Secret Sharing
- An example application: Proactive Schnorr's Signatures
- Conclusions

Proactive Secret Sharing (PSS) Scheme

Setup:

dealer establishes a $(t+1)$-*out*-n sharing of a secret s

Goal:

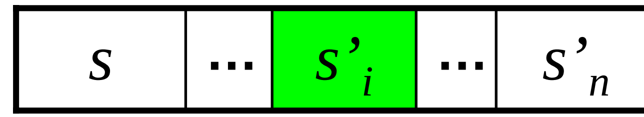
In every phase,

servers compute a ***fresh*** $(t+1)$-*out*-n sharing of s

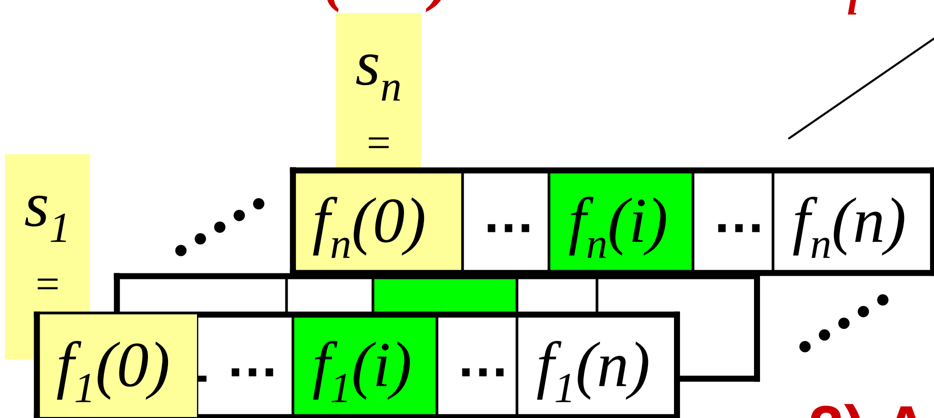
→ protects secret s from t-limited mobile adversary

PSS Implementation

Previous solutions [CKPS02]:



1) Every server i re-shares its current $(t+1)$-out-n share s_i



3) sum
- over agreed-on sharings
- use Lagrange coefficients

2) Agree on sub-set of terminating re-sharing protocols

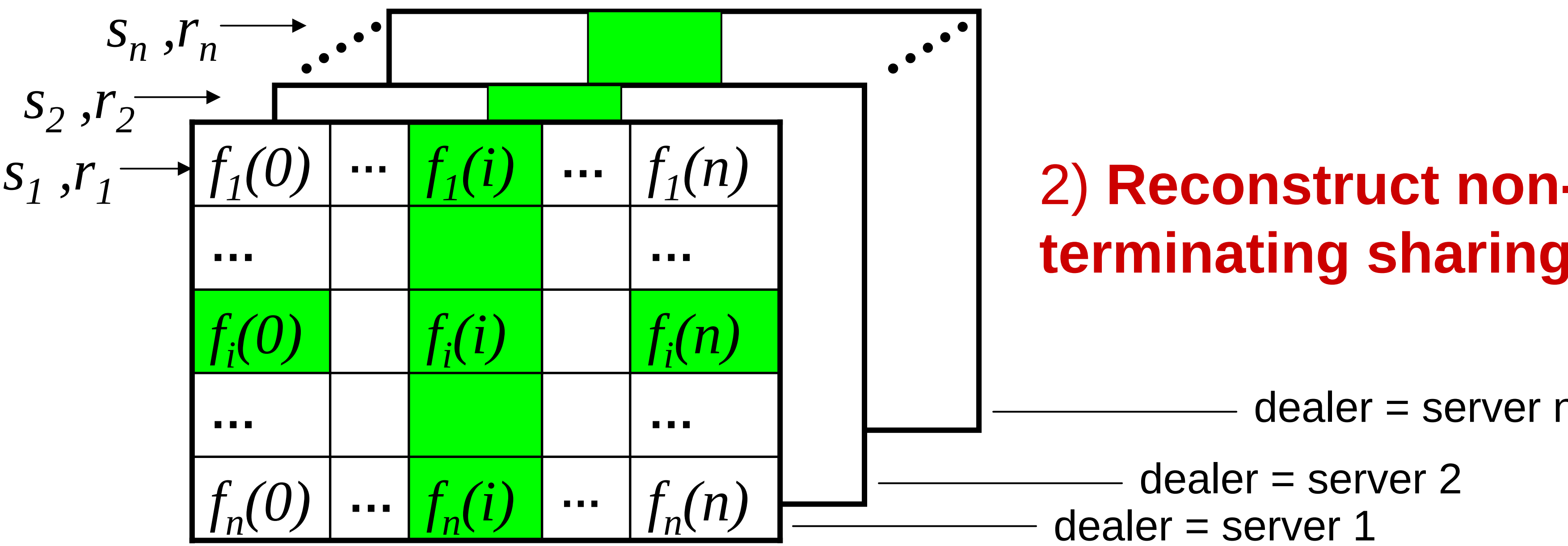
PSS Implementation

Now: Refresh hybrid sharing

1) reshare current ***additive*** share **using hybrid share and reconstructable randomness**

s'_1	…	s'_{1i}	…	s'_{1i}
…				…
s'_{i0}		s'_{ii}		s'_{in}
…				…
s'_{n0}	…	s'_{ni}	…	s'_{nn}

3) Sum **componentwise**



2) **Reconstruct non-terminating sharings**

Proactive Joint Random Sharing (JRS) Scheme

Goal:

In every phase,

servers can repeatedly compute $(t+1)$-*out*-n sharings of ***random values*** unkown to the adversary

Implementation:

- based on Hybrid Secret Sharing combined with Reconstructible Proactive Randomness
- works as refresh in Proactive Secret Sharing

Outline of the talk

- Introduction to proactive cryptosystems
- An overview of the proposed construction
- Protocols
  - Hybrid Secret Sharing
  - Reconstructible Proactive Pseudorandomness
  - Proactive Secret Sharing and
    Joint Random Secret Sharing
- **An example application: Proactive Schnorr's Signatures**
- **Conclusions**

Schnorr's Signatures [Schnorr'91]

Setup:

p a large prime

$<g>$ multiplicative subgroup of $\mathbf{Z}^*_p$, generated by g, of order q such that $q | p\text{-}1$

H – a hash function

Signatures:

secret key: x (randomly drawn from $\mathbf{Z}_q$)

public key: $y = g^x$

a signature of a message m is (R, S), where

r is random from $\mathbf{Z}_q$,
$R = g^r \bmod p$,
$S = r + \mathrm{H}(m||R)\, x \bmod q$

to verify a signature (R, S) of m check

$$g^S = R\, y^{\mathrm{H}(m||R)} \bmod p$$

Proactive Schnorr's Signatures

Maintaining the secret key:

run PSS scheme → in every phase, every server P_i receives a fresh $(t+1)$-out-n share x_i of the secret x

Signing message m:

choosing r:

run the JRSS protocol → every server P_i receives a share r_i of a random value r

compute $R = g^r \bmod p$:

every server broadcasts g^{r_i}

from $t+1$ such values, compute $R = \prod (g^{r_i})^{\lambda_i}$

compute $S = r + \mathrm{H}(m||R)\, x \bmod q$:

every server broadcasts $s_i = r_i + \mathrm{H}(m||R)\, x_i \bmod q$

from $t+1$ such values, compute $S = \sum s_i \lambda_i$

Conclusions

Asynchronous Proactive Secret Sharing and
Joint Random Secret Sharing

- do not need agreement
- have efficient worst-case complexity

→ large class of DL-based cryptosystems can be efficiently proactivized (asynchronously)

Open problems

can we do the same for Byzantine adversary?