Practical Forward Secure Sequential Aggregate Signatures

Di Ma

Computer Science Department
University of California, Irvine
Mar. 19, 2008

Outline

- Introduction
- New Constructions
- Evaluation
- Conclusion

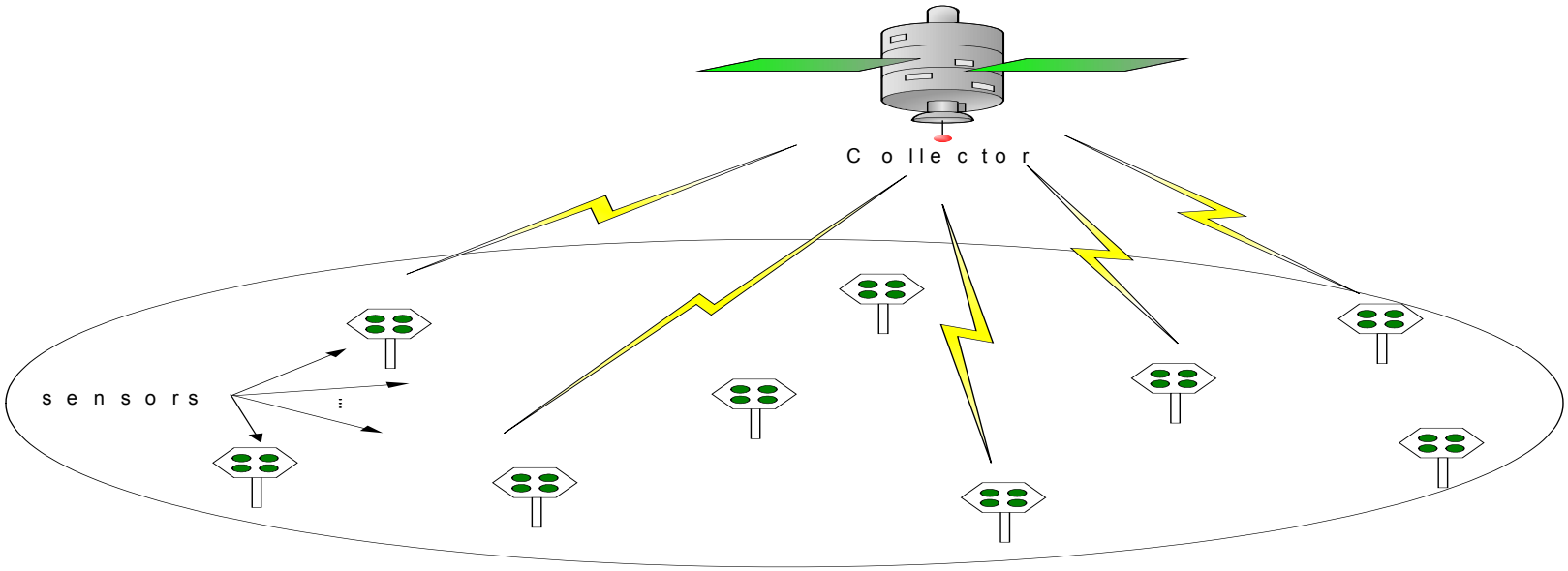
Forward-Secure Sequential Aggregate (*FssAgg*) Signatures

- Allow a signer to combine multiple signatures generated in different time periods into a single constant-size signature sequentially
  - Aggregation of forward secure signatures of the same signer
  - Aggregation is done sequentially with time
- Applications where both forward security and storage/communication efficiency are needed
  - Verifying the aggregate simultaneously verifies every component signature
  - Compromise of current key does not invalidate the authenticity of pre-compromise data

Motivating Example

Unattended sensors:

- Sensors do not network
- Sensors unable to communicate to sink at will (in real time)
- Collect data and wait for the sink



Prob: how to authenticate data accumulated during multiple intervals

Two Issues

- Issue 1: Threat of Sensor Compromise
  - Sensors are not tamper resistant devices
  - Adversarial and unattended environment
- Issue 2: Storage and Communication Overhead
  - Sensors have limited on-board storage
  - Wireless communication is very expensive

Existing Tools …

- Normal MAC/Signatures
  - Provide authentication only when key is kept secure
- Forward Secure MAC/Signatures
  - Require generating one authentication tag per message
  - *Too much storage and bandwidth overhead*

Definition

Component algorithms:

1. $Kg(k,T) \rightarrow (k_0, VK)$
   Generate the initial private k_0 and the public key VK

2. $Asig(m_i, \sigma_{1,i-1}, k_i) \rightarrow \sigma_{1,i}$
   Generate a FssAgg signature $\sigma_{1,i}$ on current message m_i
   the aggregate-so-far FssAgg signature $\sigma_{1,i-1}$ with current signing key k_i

   And invoke **3.** $Upd(k_i) \rightarrow k_{i+1}$
   Generate k_{i+1} from k_i

4. $Aver(m_1,...,m_i, \sigma_{1,i}, VK) \rightarrow T / F$
   Verify a FssAgg signature with the verification key, accept or reject.

Properties

- **Correctness**:
  - any FssAgg signature produced by *Asig* must be accepted by *Aver*

- **Forward Secure Aggregate Unforgeability:**
  - append-only

Prior Scheme

Ma/Tsudik. IEEE S&P 2007

- BLS-based FssAgg signature

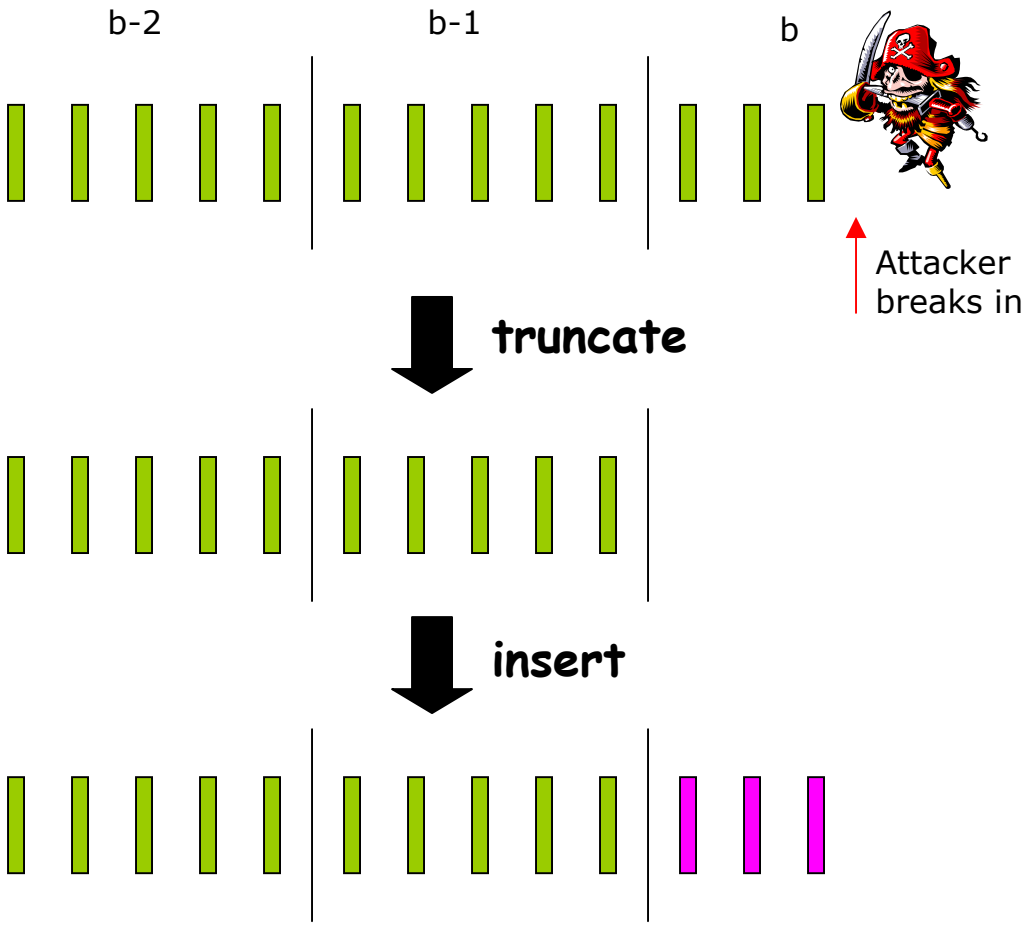
  Boneh/Lynn/Shacham. Asiacrypt 2001
  Boneh/Gentry/Lynn/Shacham, Eurocrypt 2003

  - Key update: hash function (public key remains the same)
  - Aggregation of forward-secure BLS signatures: multiplication

Performance

- Signer-friendly
  - Constant private key and signature size
  - Efficient signing and key update
- Not verifier-friendly
  - Public key size - O(*T*)
  - Costly pairing operations in verification

If key is not updated on a per message basis …



New Constructions

- Based on existing forward secure signatures
- BM-FssAgg signature
  - Bellare/Miner, Crypto 1999
- AR-FssAgg signature
  - Abdalla/Reyzin, Asiacrypt 2000
- Both are proven secure assuming the intractability of factoring a large Blum integer.

Overview of Bellare-Miner Fss Scheme

- Based on:
  - Fiat-Shamir [Crypto'86] signature scheme
  - Squaring in QR_n is a one-way permutation

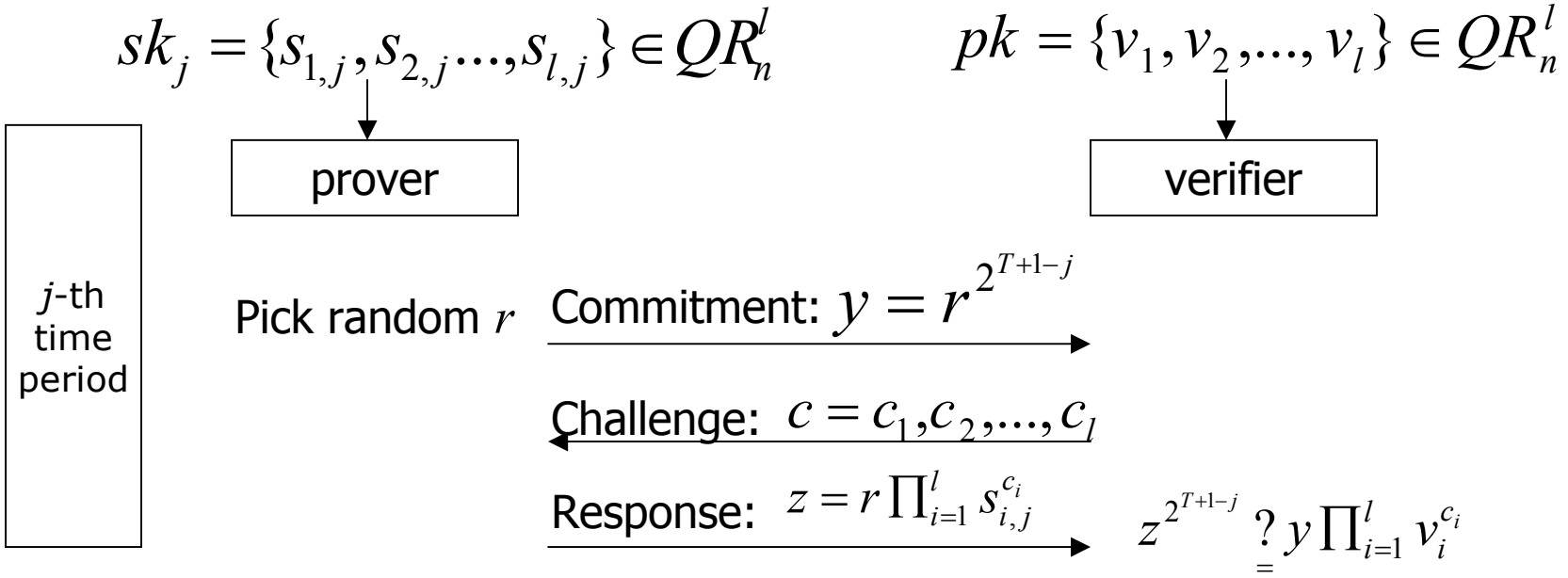
$$n = p \times q; \quad p, q : l/2\text{-bits primes}; \quad p = q = 3 \bmod 4$$

- Public key: $v_1, v_2, \ldots, v_l \in QR_n$
- Private key in *j*-th interval: $s_{1,j}, s_{2,j}, \ldots, s_{l,j} \in QR_n$

$$\{v_1, v_2, \ldots, v_l\} = \{s_{1,j}^{2^{T+1-j}}, s_{2,j}^{2^{T+1-j}}, \ldots, s_{l,j}^{2^{T+1-j}}\}$$

- Signature in the j-th interval: a non-interactive proof of knowledge of 2^{T-j+1}-th root of the public key
- Key update through squaring: $s_{i,j+1} \leftarrow s_{i,j}^2$

Overview of Bellare-Miner Fss Scheme



- Replace random c with: $c = H(j, y, m)$ to turn it into a signature scheme
- Signature format: $\sigma = (z, c)$

BM Signature Aggregation

- How to aggregate BM signatures $\sigma_1 = (z_1, c_1)$, $\sigma_2 = (z_2, c_2)$,…, $\sigma_t = (z_t, c_t)$, each from a different time period?
  - No way to aggregate c_i – hash outputs
- General idea:
  - Work on another signature format: $\sigma = (z, y)$
  - Recall in j-th time period, $r_j^{2^{T+1-j}} = y$
  - Squaring in QR_n is one-way
  - Fix y and evolve r_i through squaring
    - Pick random r_0, $r_1 = r_0^2 \rightarrow r_2 = r_1^2 \rightarrow \ldots \rightarrow r_i^{2^{T+1-i}} = y$
    - Now y is fixed, can be moved as part of the public key
  - Aggregate z through multiplication

The BM FssAgg Signature Scheme

1. FssAgg.Kg: pick randomly $s_{1,0}, s_{2,0} \ldots, s_{l,0}, r_0 \leftarrow Z_n^*$

Set $sk_1 = \{s_{1,0}^2, s_{2,0}^2, \ldots, s_{l,0}^2, r_0^2\}$

$$pk = \{v_1, v_2, \ldots, v_l, y\} = \{1/s_{1,0}^{2^{T+1}}, 1/s_{2,0}^{2^{T+1}}, \ldots, 1/s_{l,0}^{2^{T+1}}, 1/r_0^{2^{T+1}}\}$$

2. FssAgg.Asig: given new message m_t, current key sk_t, and aggregate-so-far signature $\sigma_{1,t-1}$

Compute: 1) $c_1 c_2 \ldots c_l \leftarrow H(t, y, m_t)$

2) $z_t \leftarrow r_t \prod_{j=1}^{l} s_{j,t}^{c_j}$

3) $\sigma_{1,t} \leftarrow \sigma_{1,t-1} \bullet z_t$

3. FssAgg.Upd: $sk_{t+1} = \{s_{1,t}^2, s_{2,t}^2, \ldots, s_{l,t}^2, r_i^2\}$

The BM FssAgg Signature Scheme

4. FssAgg.Aver: to verify $\sigma_{1,t}$ over messages $m_1, m_2, \ldots, m_t$ with public key $pk = \{v_1, v_2, \ldots, v_l, y\}$

For j = t to 1

-- Compute $c_1 c_2 \ldots c_l \leftarrow H(i, y, m_j)$

-- if j = t , compute $\sigma' \leftarrow \sigma_{1,t}^{2^{T+1-j}} \bullet y \bullet \prod_{i=1}^{l} v_i^{c_i}$

-- else compute $\sigma' \leftarrow \sigma'^2 \bullet y \bullet \prod_{i=1}^{l} v_i^{c_i}$

If $\sigma' = 1$ return 1; else return 0.

Performance Evaluation

Asymptotic Evaluation of Signature-based FssAgg Schemes

Parameters	**BLS-FssAgg**	**BM-FssAgg**	**AR-FssAgg**
Signature Size	O(1)	O(1)	O(1)
Secret Key Size	O(1)	O(l)	O(1)
Upd Time	O(1)	O(1)	O(l)
Asig Time	O(1)	O(1)	O(l)
Public Key Size	O(T)	O(l)	O(1)
Aver Time	O(T)	O(T)	O(lT)

bandwidth →

signer storage →

signer computation →

verifier storage →

verifier computation →

l: security parameter with practical value of 160

T: maximum number of time periods

Operation Timing

- Intel dual-core 1.73GHz Laptop with 1GB RAM running Linux
- Cryptographic libraries: NTL and PBC
- BM- and AR-FssAgg security parameters: k=1024, l=160
- BLS-FssAgg security parameters: $|q|$=512, $|p|$=160

		BM-based	AR-based	BLS-based
Signer Computation (per msg)	*Asig*	2.09	4.39	30
	Upd	3.46	7.27	0.002
	total	5.55	11.66	30.00
Verifier Computation	T=100	211.97	810.88	3.30×10^3
	T=1000	2.13×10^3	8.16×10^3	29.3×10^3
	T=10000	21.35×10^3	80.84×10^3	330.72×10^3

Other Applications

- Secure Logging
- Secure Camera
- Mobile Agent

Applications with a need to authenticate data generated and stored on an unattended and untrusted device

Conclusion

- Applications that look for efficient FssAgg signature schemes
- Proposed two practical FssAgg signature schemes
  - Trivial
  - Too simple (if simplicity is a shortcoming )
  - No much cryptographic contribution
- Expect more efficient schemes coming up from the crypto research community

Thanks!



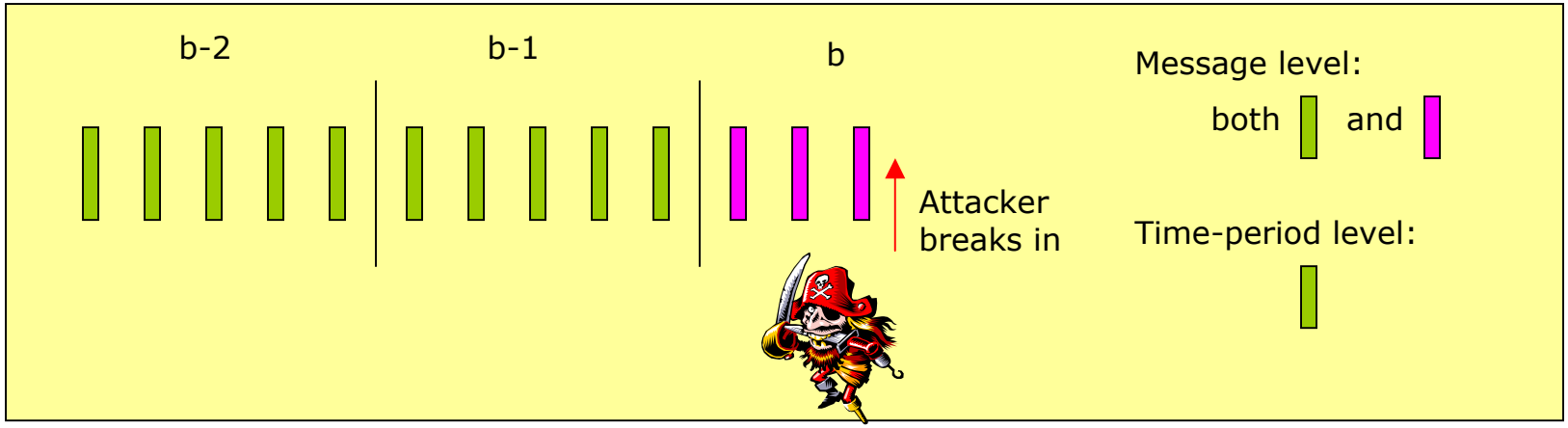
Extension: Supporting Multiple Messages

- Select *m* random numbers: r_i , *i*=1,…,*m*
- Calculate *m* commitments: $y_i = r_i^{2^{T+1}}$, *i*=1,…,*m*
- Use y_i and its corresponding 2^{T-j+1} root to sign the *i*-th message in the *j*-th interval

Have used the extension to achieve immutability in the context of FssAgg signatures

Quality of Forward Security (QFS)

- QFS: at what level, forward security can be provided.
- Two levels of QFS when key update is not done on a per message basis:
  - Message Level Forward Security
    - Trustworthy of messages generated before the compromise time point remains
  - Time-period Level Forward Security
    - Trustworthy of messages generated before the compromise time period remains.



Quality of Forward Security (QFS)

- The FssAgg MAC scheme provides message level QFS
- The BLS-based FssAgg signature scheme provides time-period level QFS
- Working with a FssAgg scheme with time-period level QFS:
  - To achieve strong security, key update should be invoked immediately after aggregate signature generation
  - In practice, multiple messages generated in the same time period can be treated as on whole BIG message – less computation