Concurrent Signatures

Liqun Chen[1], Caroline Kudla[2*] and Kenneth G. Paterson[2†]

liqun.chen@hp.com, {c.j.kudla, kenny.paterson}@rhul.ac.uk

[1]Hewlett-Packard Laboratories, Bristol, UK

[2]Information Security Group

Royal Holloway, University of London, UK

[*]This author supported by Hewlett-Packard Laboratories.

[†]This author supported by the Nuffield Foundation NUF-NAL02.

Contents of this Talk

Introduction to Concurrent Signatures

A concurrent signature scheme is a signature scheme where:

- Users can initially exchange *non-binding* signatures that are somehow linked, and
- All signatures can concurrently be converted to full *binding* signatures.
- Either no signatures are binding, or all signatures are.

The Building Blocks

Ambiguous signatures:

- Could have been produced by either of two parties,
- Can convince other party but no-one else,
- E.g. Two party ring signatures, designated verifier signatures.

Keystone:

- Seed for a *keystone fix*,
- Release of keystone removes ambiguity.

How do Concurrent Signatures Work?

Suppose entities A and B wish to exchange signatures on messages M_A and M_B respectively.

$\mathbf{A}$		$\mathbf{B}$
$k \rightarrow$ keystone fix f		
$f \rightarrow$ ambig. signature σ_A	$\xrightarrow{\sigma_A}$	

How do Concurrent Signatures Work?

Suppose entities A and B wish to exchange signatures on messages M_A and M_B respectively.

A		**B**
$k \rightarrow$ keystone fix f		
$f \rightarrow$ ambig. signature σ_A	$\xrightarrow{\sigma_A}$	
		B verifies σ_A
	$\xleftarrow{\sigma_B}$	$f \rightarrow$ ambig. signature σ_B

How do Concurrent Signatures Work?

Suppose entities A and B wish to exchange signatures on messages M_A and M_B respectively.

A		**B**
$k \longrightarrow$ keystone fix f		
$f \longrightarrow$ ambig. signature σ_A	$\xrightarrow{\sigma_A}$	
		B verifies σ_A
	$\xleftarrow{\sigma_B}$	$f \longrightarrow$ ambig. signature σ_B
A verifies σ_B		
A reveals k	$\xrightarrow{k}$	

The pairs $\langle \sigma_A, k \rangle$ and $\langle \sigma_B, k \rangle$ are called *concurrent signatures*.

Fair Exchange of Signatures

Fair exchange of signatures allow mutually distrustful parties to exchange signatures in a *fair* way.

Fair means: Either each party obtains the other's signature, or neither party does.

Two main approaches to fair exchange of signatures:

1. Timed release of signatures,
2. Solutions involving a trusted third party.

Applications of Concurrent Signatures

The Problem: A may never reveal the keystone to B.

But: Same keystone validates both A and B's signatures, so either signature with keystone validates other signature.

Existing mechanisms can guarantee delivery of keystone to B. So concurrent signatures are applicable when:

Applications of Concurrent Signatures

The Problem: A may never reveal the keystone to B.

But: Same keystone validates both A and B's signatures, so either signature with keystone validates other signature.

Existing mechanisms can guarantee delivery of keystone to B. So concurrent signatures are applicable when:

- A cannot withhold the keystone because she needs it to obtain a service from B. (E.g. Computer Depot)

Applications of Concurrent Signatures

The Problem: A may never reveal the keystone to B.

But: Same keystone validates both A and B's signatures, so either signature with keystone validates other signature.

Existing mechanisms can guarantee delivery of keystone to B. So concurrent signatures are applicable when:

- A cannot withhold the keystone because she needs it to obtain a service from B. (E.g. Computer Depot)
- A cannot keep B's signature private in the long term (E.g. Credit card system).

Applications of Concurrent Signatures

The Problem: A may never reveal the keystone to B.

But: Same keystone validates both A and B's signatures, so either signature with keystone validates other signature.

Existing mechanisms can guarantee delivery of keystone to B. So concurrent signatures are applicable when:

- A cannot withhold the keystone because she needs it to obtain a service from B. (E.g. Computer Depot)
- A cannot keep B's signature private in the long term (E.g. Credit card system).
- A single third party C will verify both signatures (E.g. Politicians and press)

Technical Approach

Ambiguous signature: Could be created by either of two parties.

Keystone: Converts ambiguous signature into full binding signature.

Approach:

Technical Approach

Ambiguous signature: Could be created by either of two parties.

Keystone: Converts ambiguous signature into full binding signature.

Approach:

1. Ring signatures have desired ambiguity properties.

Technical Approach

Ambiguous signature: Could be created by either of two parties.

Keystone: Converts ambiguous signature into full binding signature.

Approach:

1. Ring signatures have desired ambiguity properties.
2. Rivest et al. [RST01]: Signer can prove authorship by choosing certain bits *pseudorandomly*, and later reveal the seed used.

Technical Approach

Ambiguous signature: Could be created by either of two parties.

Keystone: Converts ambiguous signature into full binding signature.

Approach:

1. Ring signatures have desired ambiguity properties.
2. Rivest et al. [RST01]: Signer can prove authorship by choosing certain bits *pseudorandomly*, and later reveal the seed used.

We use the ring signature scheme of Abe et al. [AOS02] (an adaptation of Schnorr signature scheme [S91]).

We call our seed a *keystone*, and use a *cryptographic hash function* to create the keystone fix from the keystone.

Definition of Scheme

A concurrent signature scheme is a digital signature scheme comprised of the following algorithms:

SETUP: On input a security parameter l, outputs the public parameters, a function KGEN, and the participants' public and private keys.

ASIGN: A probabilistic algorithm that produces an ambiguous signature on a message M.

AVERIFY: An algorithm which verifies an ambiguous signature.

VERIFY: An algorithm which takes a keystone and an ambiguous signature, verifies the ambiguous signature, and tests whether the keystone is valid.

The Concrete Scheme (1)

SETUP: On input a security parameter l:

- Select two large primes p and q s.t. $q|p-1$.
- Select an element $g \in \mathbb{Z}_p^*$ of order q.
- Set the message and keystone spaces $\mathcal{M}, \mathcal{K}$=$\{0,1\}^*$, the signature and keystone fix spaces $\mathcal{S}, \mathcal{F} \equiv \mathbb{Z}_q$.
- Select two cryptographic hash functions $H_1, H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q$ and set KGEN=H_1.
- Select private keys $x_i \in_R \mathbb{Z}_q$ and set the public keys $X_i = g^{x_i} \bmod p$.

The Concrete Scheme (2)

ASIGN: On input $\langle X_i, X_j, x_i, h_2, M\rangle$, pick random $t \in \mathbb{Z}_q$ and compute:

$$h = H_2(g^t {X_j}^{h_2} \bmod p||M),$$

$$h_1 = h - h_2 \bmod q \ , \ \ s = t - h_1 x_i \bmod q.$$

Output $\sigma = \langle s, h_1, h_2\rangle$.

AVERIFY: On input $\langle \sigma, X_i, X_j, M\rangle$ where $\sigma = \langle s, h_1, h_2\rangle$, verify the equation

$$h_1 + h_2 = H_2(g^s X_i^{h_1} X_j^{h_2} \bmod p \ ||M) \quad \bmod q$$

Output accept or reject.

The Concrete Scheme (3)

VERIFY: On input $\langle k, S\rangle$ where $k \in \mathcal{K}$, $S = \langle \sigma, X_i, X_j, M\rangle$, check if KGEN($k$)= h_2.

If not, output reject, otherwise run AVERIFY(S).

Security Model

Security is defined via the following notions:

Correctness: AVERIFY accepts signatures produced by ASIGN.

Unforgeability: The adversary should not be able to create (ambiguous) signatures without the appropriate private key.

Ambiguity: The adversary should not be able to distinguish which of two possible signers created an ambiguous signature.

Fairness: If two ambiguous signatures use the same keystone fix f, then a keystone will either convert both signatures into full signatures, or neither.

Unforgeability Game

We define existential unforgeability of a concurrent signature scheme under a chosen message attack using the following game between an adversary E and a challenger C.

Setup: C runs SETUP for a security parameter l. E is given public parameters and the public keys $\{X_i\}$. C retains the private keys $\{x_i\}$.

Queries: E can make the following queries to C: KGen Queries, KReveal Queries, ASign Queries, and Private Key Extract Queries.

Unforgeability Definition

Output: Finally E outputs a tuple $\sigma = \langle s, h_1, f \rangle$ with public keys X_c, X_d, and message $M \in \mathcal{M}$. The adversary wins if AVERIFY$(\langle s, h_1, f \rangle, X_c, X_d, M)$= accept, and if either:

1. No ASign query on either $\langle X_c, X_d, f, M \rangle$ or $\langle X_d, X_c, h_1, M \rangle$ was made by E, and no Private Key Extract query was made on either X_c or X_d.
2. No ASign query on $\langle X_c, X_i, f, M \rangle$ was made for any $X_i \neq X_c, X_i \in \mathcal{U}$, no Private Key Extract query on X_c was made, and either f was a previous output from a KGen query or E produces a keystone k such that $f =$ KGEN(k).

Definition: A concurrent signature scheme is *existentially unforgeable under a chosen message attack* if the probability of success of any polynomially bounded adversary in the above game is negligible.

Security Results

Theorem: *Our concrete concurrent signature scheme is existentially unforgeable under a chosen message attack in the random oracle model, assuming the hardness of the discrete logarithm problem.*

Security results are also proved for the ambiguity and fairness properties of the concrete scheme.

Extensions and Open Problems

- Extension to the multi-party case with appropriate model of fairness.

Extensions and Open Problems

- Extension to the multi-party case with appropriate model of fairness.
- To investigate methods whereby the revelation of the keystone does not depend entirely on the initial signer.

Conclusions

- Introduced the notion of concurrent signatures and compared it to previous work,
- Discussed applications for concurrent signatures,
- Presented a concrete concurrent signature scheme,
- Related the security of the concrete scheme to the hardness of the discrete logarithm problem in an appropriate security model.

Concurrent Signatures

Liqun Chen[1], Caroline Kudla[2*] and Kenneth G. Paterson[2†]

liqun.chen@hp.com, {c.j.kudla, kenny.paterson}@rhul.ac.uk

[1]Hewlett-Packard Laboratories, Bristol, UK

[2]Information Security Group

Royal Holloway, University of London, UK

[*]This author supported by Hewlett-Packard Laboratories.

[†]This author supported by the Nuffield Foundation NUF-NAL02.