CS 758: Cryptography/Network Security

Douglas R. Stinson

David R. Cheriton School of Computer Science
University of Waterloo

Spring 2016

Table of Contents

CS 758: Cryptography / Network Security

- offered in the Winter term, 2015, by Doug Stinson
- my office: DC 3522
- my email address: `dstinson@uwaterloo.ca`
- my web page:

  `http://www.math.uwaterloo.ca/~dstinson/`

  provides a link to the course web page:

  `http://www.math.uwaterloo.ca/~dstinson/CS_758/S16/CS758.html`

Objectives/Prerequisites

- basic cryptography concerns secure communication between two parties, while in this course we are interested in cryptographic protocols in multiuser/network context
- prerequisites: a previous course in cryptography (e.g. C&O 487, Applied Cryptography) is helpful but not required
- mathematical background: basic complexity theory, elementary number theory, algebra (finite groups, finite fields, linear algebra), probability (random variables), combinatorics
- This is not a math course, but it has significant mathematical content. **If you are math-phobic, this course is not for you.**

Course Requirements

- Students' grades will be based on some combination of assignments and a project.
- I encourage students to work in groups of two (or three) for the project.
- The project will include a written component as well as a presentation in class.
- The project will involve
  - preparing a report on a recent research paper on a topic related to the course material, and/or
  - implementing and analyzing one or more protocols on a topic related to the course material.

Course Outline

- Review of cryptographic primitives and their applications to information security, and notions of cryptographic security. Discussion of public-key encryption, secret-key encryption, message authentication, signature schemes, and hash functions.
- Techniques for entity authentication. Passwords, challenge-response, identification schemes (e.g., Fiat-Shamir, Guillou-Quisquater), general techniques for zero-knowledge proofs for NP-complete languages.

Course Outline (cont.)

- Protocols for key establishment, transport, agreement and maintenance. Online key distribution using a trusted server (Kerberos). Public-key techniques, including Diffie-Hellman key agreement, man-in-the-middle attacks, STS and forward secrecy. Unconditionally secure key distribution, including the Blom scheme and combinatorial key distribution patterns.
- Cryptography in a multi-user setting. Secret sharing schemes (including Shamir threshold schemes and schemes for general access structures). Conference key distribution and broadcast encryption. Copyright protection techniques and tracing schemes.

Introduction to Cryptography and Security

In the rest of this module, we discuss the following:

- goals of cryptography
- cryptographic tools (primitives)
- mathematical background
- a formal model for security

Table of Contents

Goals of Cryptography

confidentiality

Confidentiality (or **secrecy**) means that data cannot be understood by an unauthorized party.

data integrity

Data integrity means that data cannot be modified by an unauthorized party.

data origin authentication

Data origin authentication is achieved when it can be verified that data was transmitted by a particular source.

entity authentication

Entity authentication (or **identification**) refers to the verification of the identity of a person, computer or other device.

Goals of Cryptography (cont.)

non-repudiation

Non-repudiation occurs when it is impossible for someone to deny having transmitted a message that, in fact, they did transmit.

access control

Access control refers to the restriction of electronic or physical access to authorized parties.

anonymity

Anonymity refers to the anonymous transmission of data, so that the origin cannot be determined.

Cryptographic Tools

encryption schemes

Encryption schemes are used to achieve confidentiality.

signature schemes

Signature schemes are used to "sign" data. A signature helps to ensure data integrity and data origin authentication, and it can also provide non-repudiation.

message authentication codes

A message authentication code provides data integrity.

cryptographic hash functions

A hash function shrinks a long string into a short, random-looking string. They are used to provide unpredictable redundancy in data. They are also used for key derivation.

Cryptographic Tools (cont.)

key agreement protocols

A key agreement protocol is used to establish a common secret key known to two or more specified parties. Usually this key is to be subsequently used for another cryptographic purpose such as symmetric-key encryption or message authentication.

identification schemes

An identification scheme provides entity authentication.

pseudorandom number generators

Pseudorandom number generators expand a small, truly random, seed into a long string of bits that cannot be distinguished from random bits. Pseudorandom number generators are used in many cryptographic contexts, for example, in the generation of keys.

Tools and their Usage of Keys

A short summary of cryptographic tools and their usage of keys is provided in the following table. A check mark ✓ indicates that the given algorithm and key combination is feasible.

	keys		
scheme	public/private?	secret?	no key?
encryption scheme	✓	✓	
signature scheme	✓	✓	
MAC		✓	
hash function			✓
key agreement scheme	✓	✓	
identification scheme	✓	✓	

Secure Socket Layer

client		server
	$\xrightarrow{\text{I'm Alice}}$	
	$\xleftarrow{\text{I'm Bob, Inc.}}$	
	$\xleftarrow{\mathbf{PK}, \mathrm{sig}_{\mathbf{CA}}(\mathbf{PK})}$	
verify **PK**		
generate **MS**	$\xrightarrow{y = e_{\mathbf{PK}}(\mathbf{MS})}$	
$K_1, K_2 = h(\mathbf{MS})$		$\mathbf{MS} = d_{\mathbf{PK}}(y)$ $K_1, K_2 = h(\mathbf{MS})$

Cryptosystem

A **cryptosystem** is a five-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where the following conditions are satisfied:

1. $\mathcal{P}$ is a finite set of possible **plaintexts**
2. $\mathcal{C}$ is a finite set of possible **ciphertexts**
3. $\mathcal{K}$, the *keyspace*, is a finite set of possible **keys**
4. For each $K \in \mathcal{K}$, there is an **encryption rule** $e_K \in \mathcal{E}$ and a corresponding **decryption rule** $d_K \in \mathcal{D}$. Each $e_K : \mathcal{P} \to \mathcal{C}$ and $d_K : \mathcal{C} \to \mathcal{P}$ are functions such that $d_K(e_K(x)) = x$ for every plaintext element $x \in \mathcal{P}$.

Public-key vs Secret-key Cryptosystems

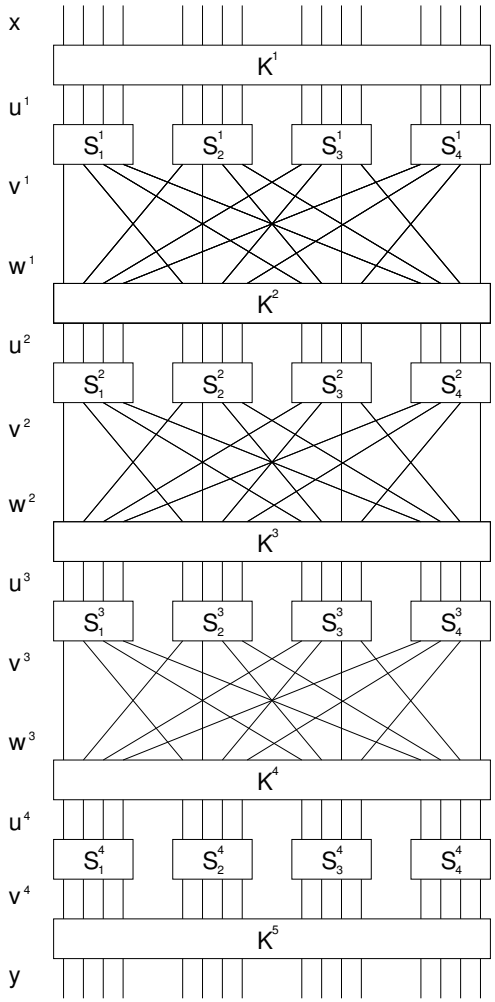
- in a secret-key cryptosystem, K is known to both Alice and Bob:

Alice		Bob
K		K
$y = e_K(x)$	$\xrightarrow{\quad y \quad}$	$x = d_K(y)$

- in a public-key cryptosystem, K is known only to Bob and e_K is public:

Alice		Bob
e_k		K
$y = e_K(x)$	$\xrightarrow{\quad y \quad}$	$x = d_K(y)$

A Substitution-Permutation Network



The Advanced Encryption Standard (AES)

AES has a block length of 128 bits, and it supports key lengths of 128, 192 and 256 bits. The number of rounds, Nr, depends on the key length: $\mathrm{Nr} = 10$ if the key length is 128 bits; $\mathrm{Nr} = 12$ if the key length is 192 bits; and $\mathrm{Nr} = 14$ if the key length is 256 bits.

1. Given a plaintext x, initialize State to be x and perform *AddRoundKey*, which x-ors the RoundKey with State.
2. For each of the first $\mathrm{Nr} - 1$ rounds, perform a substitution operation called *SubBytes* on State using an S-box; perform a permutation *ShiftRows* on State; perform an operation *MixColumns* on State; and perform *AddRoundKey*.
3. Perform *SubBytes*; perform *ShiftRows*; and perform *AddRoundKey*.
4. Define the ciphertext y to be State.

AES States

All operations in *AES* are byte-oriented operations, and all variables used are considered to be formed from an appropriate number of bytes. The plaintext x consists of 16 bytes, denoted $x_0, \ldots, x_{15}$. State is represented as a four by four array of bytes, initialized as follows:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$\longleftarrow$

x_0	x_4	x_8	x_{12}
x_1	x_5	x_9	x_{13}
x_2	x_6	x_{10}	x_{14}
x_3	x_7	x_{11}	x_{15}

The Finite Field $\mathbb{F}_{256}$

The operation *SubBytes* performs a substitution on each byte of State independently, which involves operations in the finite field

$$\mathbb{F}_{2^8} = \mathbb{Z}_2[x]/(x^8 + x^4 + x^3 + x + 1).$$

Let *BinaryToField* convert a byte to a field element; and let *FieldToBinary* perform the inverse conversion. This conversion is done in the obvious way: the field element

$$\sum_{i=0}^{7} a_i x^i$$

corresponds to the byte

$$a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0,$$

where $a_i \in \mathbb{Z}_2$ for $0 \leq i \leq 7$.

SubBytes

Algorithm: *SubBytes*$(a_7a_6a_5a_4a_3a_2a_1a_0)$
external *FieldInv*, *BinaryToField*, *FieldToBinary*
$z \leftarrow$ *BinaryToField*$(a_7a_6a_5a_4a_3a_2a_1a_0)$
if $z \neq 0$
  then $z \leftarrow$ *FieldInv*(z)
$(a_7a_6a_5a_4a_3a_2a_1a_0) \leftarrow$ *FieldToBinary*(z)
$(c_7c_6c_5c_4c_3c_2c_1c_0) \leftarrow (01100011)$
for $i \leftarrow 0$ **to** 7
  do $b_i \leftarrow (a_i + a_{i+4} + a_{i+5} + a_{i+6} + a_{i+7} + c_i) \bmod 2$
return $(b_7b_6b_5b_4b_3b_2b_1b_0)$

ShiftRows

The operation *ShiftRows* acts on State as shown in the following diagram:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$\leftarrow$

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

MixColumns

Algorithm: *MixColumn*(c)
external *FieldMult*, *BinaryToField*, *FieldToBinary*
for $i \leftarrow 0$ **to** 3
  do $t_i \leftarrow$ *BinaryToField*$(s_{i,c})$
$u_0 \leftarrow$ *FieldMult*$(x, t_0) \oplus$ *FieldMult*$(x+1, t_1) \oplus t_2 \oplus t_3$
$u_1 \leftarrow$ *FieldMult*$(x, t_1) \oplus$ *FieldMult*$(x+1, t_2) \oplus t_3 \oplus t_0$
$u_2 \leftarrow$ *FieldMult*$(x, t_2) \oplus$ *FieldMult*$(x+1, t_3) \oplus t_0 \oplus t_1$
$u_3 \leftarrow$ *FieldMult*$(x, t_3) \oplus$ *FieldMult*$(x+1, t_0) \oplus t_1 \oplus t_2$
for $i \leftarrow 0$ **to** 3
  do $s_{i,c} \leftarrow$ *FieldToBinary*(u_i)

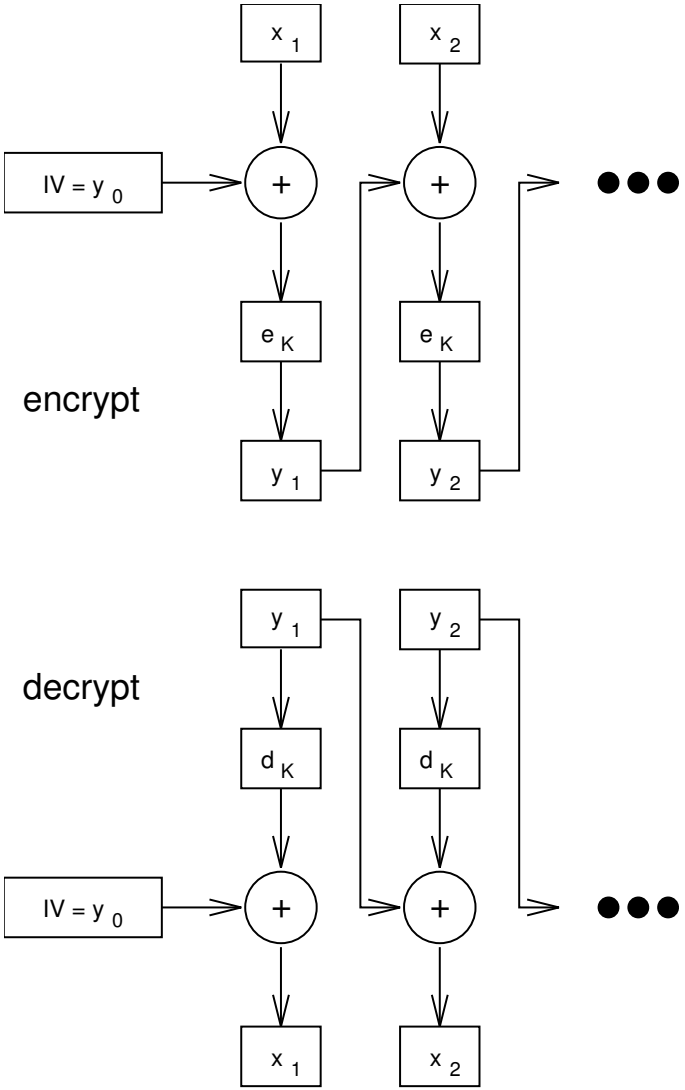
Modes of Operation

- ECB (**electronic code book**) mode corresponds to the naive use of a block cipher: given a sequence $x_1 x_2 \cdots$ of plaintext blocks (each consisting of 128 bits, in the case of the *AES*), each x_i is encrypted with the same key K, producing a string of ciphertext blocks, $y_1 y_2 \cdots$.
- In CBC (**cipher block chaining**) mode, each ciphertext block y_i is x-ored with the next plaintext block, x_{i+1}, before being encrypted with the key K. More formally, we start with an **initialization vector**, denoted by IV, and define $y_0 = \mathrm{IV}$. Then we construct $y_1, y_2, \ldots,$ using the rule

$$y_i = e_K(y_{i-1} \oplus x_i),$$

$i \geq 1$.

CBC Mode



The RSA Public-key Cryptosystem

Let $n = pq$, where p and q are large primes. Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$, and define

$$\mathcal{K} = \{(n, p, q, a, b) : ab \equiv 1 \pmod{\phi(n)}\}.$$

For $K = (n, p, q, a, b)$, define

$$e_K(x) = x^b \bmod n$$

and

$$d_K(y) = y^a \bmod n$$

$(x, y \in \mathbb{Z}_n)$. The values n and b comprise the public key, and the values p, q and a form the private key.

A Toy Example

- suppose Bob chooses primes $p = 101$ and $q = 113$
- then $n = 11413$ and $\phi(n) = 100 \times 112 = 11200$
- suppose Bob chooses public encryption exponent $b = 3533$
- then his private decryption exponent is $a = b^{-1} \bmod 11200 = 6597$
- suppose Alice wants to encrypt the plaintext $x = 9726$
- she will compute

$$y = 9726^{3533} \bmod 11413 = 5761$$

  and send y to Bob
- when Bob receives the ciphertext $y = 5761$, he computes

$$x = 5761^{6597} \bmod 11413 = 9726.$$

The Rabin Cryptosystem

Let $n = pq$, where p and q are primes. Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n{}^*$, and define

$$\mathcal{K} = \{(n, p, q)\}.$$

For $K = (n, p, q)$, define

$$e_K(x) = x^2 \bmod n$$

and

$$d_K(y) = \sqrt{y} \bmod n.$$

The value n is the public key, while p and q are the private key.

Note: there are four square roots of y modulo n.

A Toy Example

- suppose Bob chooses primes $p = 7$ and $q = 11$
- then the encryption function is

$$e_K(x) = x^2 \bmod 77$$

and the decryption function is

$$d_K(y) = \sqrt{y} \bmod 77$$

- suppose Alice encrypts the plaintext $x = 32$ to send to Bob
- the ciphertext is $y = 32^2 \bmod 77 = 23$
- the four square roots of 23 modulo 77 are $\pm 10, \pm 32 \bmod 77$
- the four possible plaintexts are $x = 10, 32, 45$ and 67

Signature Schemes

A **signature scheme** is a five-tuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$, where the following conditions are satisfied:

1. $\mathcal{P}$ is a finite set of possible **messages**
2. $\mathcal{A}$ is a finite set of possible **signatures**
3. $\mathcal{K}$, the **keyspace**, is a finite set of possible **keys**
4. For each $K \in \mathcal{K}$, there is a **signing algorithm** $sig_K \in \mathcal{S}$ and a corresponding **verification algorithm** $ver_K \in \mathcal{V}$. Each $sig_K : \mathcal{P} \rightarrow \mathcal{A}$ and $ver_K : \mathcal{P} \times \mathcal{A} \rightarrow \{\textbf{true}, \textbf{false}\}$ are functions such that the following equation is satisfied for every message $x \in \mathcal{P}$ and for every signature $y \in \mathcal{A}$:

$$ver(x, y) = \begin{cases} \textbf{true} & \text{if } y = sig(x) \\ \textbf{false} & \text{if } y \neq sig(x). \end{cases}$$

Signatures with Hash Functions

Alice		Bob
K	without hash function	$\mathbf{ver}_K$
$y = \mathbf{sig}_K(x)$	$\xrightarrow{x, y}$	$\mathbf{ver}_K(x, y) = \mathbf{true}?$
K, h	with hash function	$\mathbf{ver}_K, h$
$z = h(x)$ $y = \mathbf{sig}_K(z)$	$\xrightarrow{x, y}$	$z = h(x)$ $\mathbf{ver}_K(z, y) = \mathbf{true}?$

$h : \{0,1\}^* \to \mathcal{Z} \subseteq \mathcal{P}$ is a public hash function

RSA Signature Scheme

Let $n = pq$, where p and q are primes. Let $\mathcal{P} = \mathcal{A} = \mathbb{Z}_n$, and define

$$\mathcal{K} = \{(n, p, q, a, b) : n = pq, p, q \text{ prime}, ab \equiv 1 \pmod{\phi(n)}\}.$$

The values n and b are the public key, and the values p, q, a are the private key.

For $K = (n, p, q, a, b)$, define

$$sig_K(x) = x^a \bmod n$$

and

$$ver_K(x, y) = \textbf{true} \Leftrightarrow x \equiv y^b \pmod{n}$$

$(x, y \in \mathbb{Z}_n)$.

Primitive Elements in $\mathbb{Z}_p$

- suppose p is prime; then there exists a **primitive element** $\alpha \in \mathbb{Z}_p$
- $\{\alpha^i \bmod p : 0 \leq i \leq p-2\} = \mathbb{Z}_p \backslash \{0\}$
- the powers of α (modulo p) yield all the non-zero elements in $\mathbb{Z}_p$
- $\alpha^{p-1} \equiv 1 \pmod{p}$, and α has order $p-1$
- if $q \mid (p-1)$, then $\beta = \alpha^{(p-1)/q} \bmod p$ has order q

Primitive Elements in $\mathbb{Z}_{13}$

- 2 is a primitive element modulo 13:

$$\begin{array}{ll} 2^0 \bmod 13 = 1 & 2^1 \bmod 13 = 2 \\ 2^2 \bmod 13 = 4 & 2^3 \bmod 13 = 8 \\ 2^4 \bmod 13 = 3 & 2^5 \bmod 13 = 6 \\ 2^6 \bmod 13 = 12 & 2^7 \bmod 13 = 11 \\ 2^8 \bmod 13 = 9 & 2^9 \bmod 13 = 5 \\ 2^{10} \bmod 13 = 10 & 2^{11} \bmod 13 = 7 \end{array}$$

- the primitive elements modulo 13 are 2, 6, 7 and 11

Digital Signature Standard

Let p be a 1024-bit prime and let q be a 160-bit prime that divides $p-1$. Let $\alpha \in {\mathbb{Z}_p}^*$ be an element of order q. Let $\mathcal{P} = \{0,1\}^*$, $\mathcal{A} = {\mathbb{Z}_q}^* \times {\mathbb{Z}_q}^*$, and define

$$\mathcal{K} = \{(p, q, \alpha, a, \beta) : \beta \equiv \alpha^a \pmod{p}\},$$

where $0 \leq a \leq q-1$. The values p, q, α and β are the public key, and a is the private key.

For $K = (p, q, \alpha, a, \beta)$, and for a (secret) random number k, $1 \leq k \leq q-1$, define

$$\mathit{sig}_K(x, k) = (\gamma, \delta),$$

where

$$\begin{aligned} \gamma &= (\alpha^k \bmod p) \bmod q \quad \text{and} \\ \delta &= (\mathit{SHA\text{-}1}(x) + a\gamma)k^{-1} \bmod q. \end{aligned}$$

Digital Signature Standard (cont.)

For $x \in \{0,1\}^*$ and $\gamma, \delta \in \mathbb{Z}_q{}^*$, verification is done by performing the following computations:

$$\begin{aligned} e_1 &= \mathit{SHA\text{-}1}(x)\,\delta^{-1} \bmod q \\ e_2 &= \gamma\,\delta^{-1} \bmod q \\ \mathit{ver}_K(x,(\gamma,\delta)) = \textbf{true} &\Leftrightarrow (\alpha^{e_1}\beta^{e_2} \bmod p) \bmod q = \gamma. \end{aligned}$$

A Toy Example

Suppose we take $q = 101$ and $p = 78q + 1 = 7879$. 3 is a primitive element in $\mathbb{Z}_{7879}$, so $\alpha = 3^{78} \bmod 7879 = 170$ has order q. Suppose $a = 75$; then $\beta = \alpha^a \bmod 7879 = 4567$. Suppose Alice wants to sign the message digest *SHA-1*$(x) = 22$, and she chooses the random value $k = 50$. Then she computes

$$k^{-1} \bmod 101 = 50^{-1} \bmod 101 = 99,$$

$$\begin{aligned} \gamma &= (170^{50} \bmod 7879) \bmod 101 \\ &= 2518 \bmod 101 \\ &= 94, \end{aligned}$$

and

$$\begin{aligned} \delta &= (22 + 75 \times 94)99 \bmod 101 \\ &= 97. \end{aligned}$$

A Toy Example (cont.)

The signature $(94, 97)$ on the message digest 22 is verified by the following computations:

$$\delta^{-1} = 97^{-1} \bmod 101 = 25$$
$$e_1 = 22 \times 25 \bmod 101 = 45$$
$$e_2 = 94 \times 25 \bmod 101 = 27$$
$$(170^{45} 4567^{27} \bmod 7879) \bmod 101 = 2518 \bmod 101 = 94.$$

Hence, the signature is valid.

Hash Functions and Families

A **hash family** is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, where the following conditions are satisfied:

1. $\mathcal{X}$ is a set of possible **messages**
2. $\mathcal{Y}$ is a finite set of possible **message digests** or **authentication tags**
3. $\mathcal{K}$, the **keyspace**, is a finite set of possible **keys**
4. For each $K \in \mathcal{K}$, there is a **hash function** $h_K \in \mathcal{H}$. Each $h_K : \mathcal{X} \to \mathcal{Y}$.

$\mathcal{X}$ can be a finite or infinite set; $\mathcal{Y}$ is always a finite set. If $\mathcal{Y}$ is finite, the we assume that $|\mathcal{X}| \geq 2|\mathcal{Y}|$.

An **unkeyed hash function** is a single hash function $h : \mathcal{X} \to \mathcal{Y}$, i.e., a hash family in which there is only one possible key.

Signature Schemes vs MACs

Alice		Bob
K, h	signature scheme	$\mathbf{ver}_K, h$
$z = h(x)$ $y = \mathbf{sig}_K(z)$	$\xrightarrow{x, y}$	$z = h(x)$ $\mathbf{ver}_K(z, y) = \mathbf{true}?$
K	MAC	K
$y = h_K(x)$	$\xrightarrow{x, y}$	$y = h_K(x)?$

Iterated Hash Functions

Suppose *compress* : $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$ is a hash function (where $t \geq 1$). We construct an iterated hash function

$$h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \rightarrow \{0,1\}^\ell$$

preprocessing

Given an input string x, where $|x| \geq m+t+1$, construct a string y, using a public algorithm, such that $|y| \equiv 0 \pmod{t}$. Typically, $y = x \parallel \mathit{pad}(x)$, where *pad* is a padding function such that the mapping $x \mapsto y$ is injective. Denote

$$y = y_1 \parallel y_2 \parallel \cdots \parallel y_r,$$

where $|y_i| = t$ for $1 \leq i \leq r$.

Iterated Hash Functions (cont.)

processing

Let IV be a public initial value which is a bitstring of length m. Then compute the following:

$$\begin{aligned} z_0 &\leftarrow \mathrm{IV} \\ z_1 &\leftarrow \mathit{compress}(z_0 \parallel y_1) \\ z_2 &\leftarrow \mathit{compress}(z_1 \parallel y_2) \\ \vdots \quad & \vdots \quad \vdots \\ z_r &\leftarrow \mathit{compress}(z_{r-1} \parallel y_r). \end{aligned}$$

optional output transformation

Let $g : \{0,1\}^m \rightarrow \{0,1\}^\ell$ be a public function. Define $h(x) = g(z_r)$.

The processing step in an iterated hash function

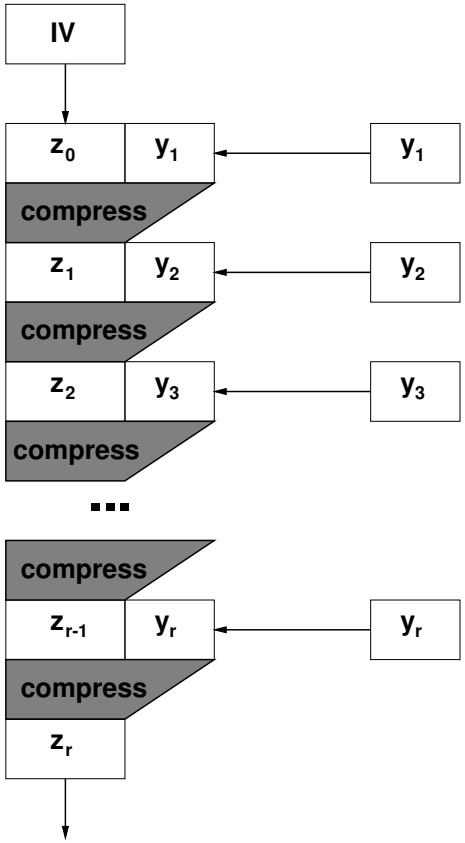


Figure: An Iterated Hash Function

Secure Hash Algorithm

SHA-1 is an iterated hash function with a 160-bit message digest. *SHA-1* is built from word-oriented operations on bitstrings, where a word consists of 32 bits (or eight hexadecimal digits). The operations used in *SHA-1* are as follows:

$X \wedge Y$	bitwise "and" of X and Y
$X \vee Y$	bitwise "or" of X and Y
$X \oplus Y$	bitwise "xor" of X and Y
$\neg X$	bitwise complement of X
$X + Y$	integer addition modulo 2^{32}
$\mathsf{ROTL}^s(X)$	circular left shift of X by s positions ($0 \leq s \leq 31$)

Secure Hash Algorithm (cont.)

Algorithm: *SHA-1-pad*(x)
$d \leftarrow (447 - |x|) \bmod 512$
$\ell \leftarrow$ the binary representation of $|x|$, where $|\ell| = 64$
$y \leftarrow x \parallel 1 \parallel 0^d \parallel \ell$

The resulting string y has length divisible by 512. Then we write y as a concatenation of n blocks, each having 512 bits:

$$y = M_1 \parallel M_2 \parallel \cdots \parallel M_n.$$

Each M_i will be written as the concatenation of 16 words:

$$M_i = W_0 \parallel W_1 \parallel \cdots \parallel W_{15}.$$

Secure Hash Algorithm (cont.)

Define the functions $f_0, \ldots, f_{79}$ as follows:

$$f_i(B, C, D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & \text{if } 0 \leq t \leq 19 \\ B \oplus C \oplus D & \text{if } 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{if } 40 \leq t \leq 59 \\ B \oplus C \oplus D & \text{if } 60 \leq t \leq 79. \end{cases}$$

Each function f_i takes three words B, C and D as input, and produces one word as output.

Algorithm: *SHA-1*(x)

initialize H_0, H_1, H_2, H_3, H_4

for $i \leftarrow 1$ **to** n

do
- **for** $t \leftarrow 16$ **to** 79
  - **do** $W_t \leftarrow \mathsf{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$
- $A \leftarrow H_0; B \leftarrow H_1; C \leftarrow H_2; D \leftarrow H_3; E \leftarrow H_4$
- **for** $t \leftarrow 0$ **to** 79
  - **do**
    - $temp \leftarrow \mathsf{ROTL}^5(A) + f_t(B, C, D) + E + W_t + K_t$
    - $E \leftarrow D; D \leftarrow C; C \leftarrow \mathsf{ROTL}^{30}(B)$
    - $B \leftarrow A; A \leftarrow temp$
- $H_0 \leftarrow H_0 + A; H_1 \leftarrow H_1 + B; H_2 \leftarrow H_2 + C$
- $H_3 \leftarrow H_3 + D; H_4 \leftarrow H_4 + E$

return $(H_0 \parallel H_1 \parallel H_2 \parallel H_3 \parallel H_4)$

CBC MACs

Suppose that $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ is a cryptosystem, where $\mathcal{P} = \mathcal{C} = \{0,1\}^t$. Let IV be the bitstring consisting of t zeroes, and let $K \in \mathcal{K}$ be a secret key. Finally, let $x = x_1 \parallel \cdots \parallel x_n$ be a bitstring of length tn (for some positive integer n), where each x_i is a bitstring of length t. We compute *CBC-MAC*$_K(x)$ as follows.

Algorithm: *CBC-MAC*(x, K)

$\mathrm{IV} \leftarrow 00 \cdots 0$ (or some other vector)

$y_0 \leftarrow \mathrm{IV}$

for $i \leftarrow 1$ **to** n

  do $y_i \leftarrow e_K(y_{i-1} \oplus x_i)$

return (y_n)

Public- vs Secret-key Cryptography

speed

Secure secret-key cryptographic protocols are much faster than correpsonding secure public-key cryptographic protocols (e.g., 100 to 1000 times faster). Here we are comparing public- vs secret-key cryptosystems; and signature schemes vs MACs.

algebraic description

Public-key cryptographic protocols are usually based on simple-to-describe mathematical functions, and their security depends on certain computational problems that are infeasible, or believed to be infeasible.

Secret-key cryptosystems tend to be based on performing a sequence of very simple operations (e.g., substitutions and permutations) in such a way that the system has a very complex algebraic description.

Public- vs Secret-key Cryptography (cont.)

key lengths

Key lengths of secure public-key cryptosystems are sometimes significantly longer than those of secure secret key cryptosystems. For example, *RSA* keys are typically taken to be at least 1024 bits in length, while an *AES* key is 128 bits long. It should be noted, however, that key lengths of elliptic curve cryptosystems are much shorter than most other public-key cryptosystems, e.g., 160 bits is a popular key length.

Public- vs Secret-key Cryptography (cont.)

utility

Public-key cryptosystems do not require that a secret key be known to two parties before the system is used, of course. Public keys can be listed in a directory or posted on a web page, for example. However, such methods of communicating public keys must also be accompanied by appropriate mechanisms to allow the public keys to be authenticated. This is done by using certificates, containing signed public keys, possibly in the context of a secure public-key infrastructure. On the other hand, the authenticity of secret keys is, in general, guaranteed by the protocol used to generate or distribute them, or assumed to have been done securely at an earlier time.

Table of Contents

Modular arithmetic

Suppose a and b are integers, and m is a positive integer. Then we write $a \equiv b \pmod{m}$ if m divides $b - a$. The phrase $a \equiv b \pmod{m}$ is called a **congruence**, and it is read as "a is **congruent** to b modulo m." The integer m is called the **modulus**.

For example, we have $11 \equiv -25 \pmod{6}$ because $6 \mid (11 - (-25))$.

We will use the notation $a \bmod m$ (without parentheses) to denote the non-negative remainder when a is divided by m. Thus, $a \equiv b \pmod{m}$ if and only if $a \bmod m = b \bmod m$. If we replace a by $a \bmod m$, we say that a is **reduced** modulo m.

For example, to compute $101 \bmod 7$, we write $101 = 7 \times 14 + 3$. Since $0 \leq 3 \leq 6$, it follows that $101 \bmod 7 = 3$. As another example, suppose we want to compute $(-101) \bmod 7$. In this case, we have $-101 = 7 \times (-15) + 4$. Since $0 \leq 4 \leq 6$, it follows that $(-101) \bmod 7 = 4$.

Modular arithmetic (cont.)

$\mathbb{Z}_n$ is is used to denote the set $\{0, \ldots, n-1\}$, which is usually equipped with the two operations $+$ and $\times$. Addition and multiplication in $\mathbb{Z}_n$ work exactly like real addition and multiplication, except that the results are reduced modulo n.

For example, suppose we want to compute 11×13 in $\mathbb{Z}_{16}$. As integers, we have $11 \times 13 = 143$. Then we reduce 143 modulo 16 as described above: $143 = 8 \times 16 + 15$, so $143 \bmod 16 = 15$, and hence $11 \times 13 = 15$ in $\mathbb{Z}_{16}$.

Finite Abelian Groups

A **finite abelian group** is a pair $G = (X, \star)$, where X is a finite set and $\star$ is a binary operation defined on X, that satisfies the following properties:

1. the operation $\star$ is **closed**, i.e., $a \star b \in X$ for any $a, b \in X$
2. the operation $\star$ is **commutative**, i.e., $a \star b = b \star a$ for any $a, b \in X$
3. the operation $\star$ is **associative**, i.e., $(a \star b) \star c = a \star (b \star c)$ for any $a, b, c \in X$
4. There is an element **id** $\in X$ called the **identity**, such that $a \star \textbf{id} = \textbf{id} \star a = a$ for any $a \in X$
5. for every $a \in X$, there exists an element $b \in X$ called the **inverse** of a, such that $a \star b = b \star a = \textbf{id}$ for any $a \in X$

The **order** of the group $G = (X, \star)$, denoted $\text{ord}(G)$, is equal to $|X|$.

Examples of Finite Abelian Groups

Let $n \geq 2$ be an integer. Then $(\mathbb{Z}_n, +)$ is an abelian group of order n, where $+$ denotes addition modulo n. The identity element is 0, and the inverse of a, usually denoted $-a$, is $(-a) \bmod n$.

Let $p \geq 2$ be a prime. Define ${\mathbb{Z}_p}^* = \mathbb{Z}_p \backslash \{0\}$. Then $({\mathbb{Z}_p}^*, \cdot)$ is an abelian group of order $p-1$, where $\cdot$ denotes multiplication modulo p. The identity element is 1, and the inverse of a, usually denoted a^{-1}, is computed using the *Extended Euclidean algorithm*.

Let $n \geq 2$ be an integer. Define

$$ {\mathbb{Z}_n}^* = \mathbb{Z}_n \backslash \{d \in \mathbb{Z}_n : \gcd(d, n) > 1\}. $$

Then $({\mathbb{Z}_n}^*, \cdot)$ is an abelian group where $\cdot$ denotes multiplication modulo n.

Examples of Finite Abelian Groups (cont.)

The identity element is 1, and the inverse of a, usually denoted a^{-1}, is computed using the *Extended Euclidean algorithm*.
The order of $(\mathbb{Z}_n{}^*, \cdot)$ is denoted $\phi(n)$ (note that $\phi(n)$ is just the number of positive integers less than n that are relatively prime to n). $\phi(n)$ can be computed from the following formula: suppose that n has prime power factorization

$$n = \prod_{i=1}^{\ell} p_i{}^{e_i}$$

(the p_i's are distinct primes and $e_i \geq 1$ for $1 \leq i \leq \ell$). Then

$$\phi(n) = \prod_{i=1}^{\ell} p_i{}^{e_i - 1}(p_i - 1) = \prod_{i=1}^{\ell} \left(p_i{}^{e_i} - p_i{}^{e_i - 1}\right).$$

Order of Group Elements

For a finite group $(X, \star)$, define the **order** of an element $a \in X$ (denoted $\text{ord}(a)$) to be the smallest positive integer m such that

$$\underbrace{a \star a \star \cdots \star a}_{m} = \textbf{id}.$$

If the group operation is multiplication, then

$$\underbrace{a \star a \star \cdots \star a}_{m}$$

is written as an exponentiation, a^m. If the group operation is addition, then the same expression is written as a multiplication, ma.
The identity element is defined to have order 1. It can be shown that the order of any $a \in X$ is a divisor of the order of the group, i.e., $\text{ord}(a) \mid \text{ord}(G)$.

Order of Group Elements (cont.)

It is also possible to show, for any $a \in X$, that the order of $b = a^i$ (where, for concreteness, we assume that the group operation is written multiplicatively) is

$$\mathsf{ord}(b) = \frac{\mathsf{ord}(a)}{\gcd(\mathsf{ord}(a), i)}.$$

For example, if $\mathsf{ord}(a) = 100$ and $b = a^{35}$, then

$$\mathsf{ord}(b) = \frac{100}{\gcd(100, 35)} = \frac{100}{5} = 20.$$

Cyclic Groups

A finite abelian group $(X, \star)$ is a **cyclic group** if there exists an element $a \in X$ having order equal to $|X|$. Such an element is called a **generator** of the group.

Let $n \geq 2$ be an integer. Then $(\mathbb{Z}_n, +)$ is a cyclic group, and 1 is a generator. Further, any element $a \in \mathbb{Z}_n$ is a generator of $(\mathbb{Z}_n, +)$ if and only if $\gcd(a, n) = 1$. The number of generators of $(\mathbb{Z}_n, +)$ is $\phi(n)$.

Let $p \geq 2$ be a prime. Then $({\mathbb{Z}_p}^*, \cdot)$ is cyclic group, and a generator of this group is called a **primitive element**.

It can be shown that $\alpha \in {\mathbb{Z}_p}^*$ is a primitive element if and only if

$$\alpha^{(p-1)/q} \not\equiv 1 \pmod{p}$$

for all primes q such that $q \mid (p-1)$. The number of generators of $({\mathbb{Z}_p}^*, \cdot)$ is $\phi(p-1)$.

Cyclic Groups (cont.)

It is known that $(\mathbb{Z}_n{}^*, \cdot)$ is cyclic group if and only if $n = 2$, 4, p^e or $2\,p^e$, where p is an odd prime and e is a positive integer.
It is simple to test whether a given element $\alpha \in \mathbb{Z}_p{}^*$ is a primitive element (where p is an odd prime) provided that the factorization of $p - 1$ is known. We illustrate with a small example, $p = 13$ and $\alpha = 2$. The factorization of 12 is $12 = 2^2 3^1$. Therefore, to verify that 2 is a primitive element modulo 13, it is sufficient to check that

$$2^6 \not\equiv 1 \pmod{13}$$

and

$$2^4 \not\equiv 1 \pmod{13}.$$

This is much faster than checking all 12 powers of α.

Quadratic Residues

Suppose p is an odd prime and a is an integer. a is defined to be a **quadratic residue** modulo p if $a \not\equiv 0 \pmod{p}$ and the congruence $y^2 \equiv a \pmod{p}$ has a solution $y \in \mathbb{Z}_p$.
It can be shown that a is a quadratic residue modulo p if and only if

$$a^{(p-1)/2} \equiv 1 \pmod{p}.$$

Every quadratic residue modulo p has exactly two square roots in $\mathbb{Z}_p$, which are negatives of each other modulo p. If $p \equiv 3 \pmod{4}$ is prime and a is a quadratic residue modulo p, then it can be shown that the two square roots of a modulo p are $\pm a^{(p+1)/4} \bmod p$.

Quadratic Residues

Suppose n is an odd integer and a is an integer. a is defined to be a **quadratic residue** modulo n if $\gcd(a, n) = 1$ and the congruence $y^2 \equiv a \pmod{n}$ has a solution $y \in \mathbb{Z}_n$. Suppose that $n > 1$ is an odd integer having factorization

$$n = \prod_{i=1}^{\ell} {p_i}^{e_i},$$

where the p_i's are distinct primes and the e_i's are positive integers. Suppose further that $\gcd(a, n) = 1$. Then a is quadratic residue modulo n if and only if

$$a^{(p_i-1)/2} \equiv 1 \pmod{p_i}$$

for all $i \in \{1, \ldots, \ell\}$. If a is quadratic residue modulo n, then there are exactly 2^ℓ solutions modulo n to the congruence $y^2 \equiv a \pmod{n}$.

Quadratic Residues (example)

As an example, suppose that $p = 43$ and $a = 13$. Note that $43 \equiv 3 \pmod 4$ is prime. It can be verified that

$$13^{21} \equiv 1 \pmod{43},$$

so 13 is a quadratic residue modulo 43. $(p+1)/4 = 11$, so the two square roots of 13 modulo 43 are

$$\pm 13^{11} \bmod 43 = \pm 23 \bmod 43 = 20, 23.$$

Extended Euclidean Algorithm

The multiplicative inverse $b^{-1} \bmod a$ exists if and only if $\gcd(a, b) = 1$. The following algorithm computes $\gcd(a, b)$, and if $\gcd(a, b) = 1$, then it also determines the value of $b^{-1} \bmod a$. These computations use the the so-called *Extended Euclidean algorithm*.

Extended Euclidean Algorithm (cont.)

Algorithm: *Multiplicative Inverse*(a, b)

$a_0 \leftarrow a; \quad b_0 \leftarrow b; \quad t_0 \leftarrow 0$

$t \leftarrow 1; \quad q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor; \quad r \leftarrow a_0 - q\, b_0$

while $r > 0$

do $\begin{cases} temp \leftarrow t_0 - q\, t; \quad t_0 \leftarrow t; \quad t \leftarrow temp \\ a_0 \leftarrow b_0; \quad b_0 \leftarrow r \\ q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor; \quad r \leftarrow a_0 - q\, b_0 \end{cases}$

if $b_0 \neq 1$

then return ("the gcd of b and a is b_0")

else $\begin{cases} t \leftarrow t \bmod a \\ \textbf{return} \ (\text{“}b^{-1} \bmod a = t\text{”}) \end{cases}$

Extended Euclidean Algorithm (example)

We compute $28^{-1} \bmod 75$. Here are the steps performed in the computation:

a_0	b_0	t_0	t	q	r
75	28	0	1	2	19
28	19	1	-2	1	9
19	9	-2	3	2	1
9	1	3	-8	9	0

Therefore,

$$28^{-1} \bmod 75 = -8 \bmod 75 = 67.$$

Chinese Remainder Theorem

Suppose $m_1, \ldots, m_r$ are pairwise relatively prime positive integers, and suppose $a_1, \ldots, a_r$ are integers. Then the system of r congruences $x \equiv a_i \pmod{m_i}$ $(1 \leq i \leq r)$ can be shown to have a unique solution modulo $M = m_1 \times \cdots \times m_r$. The *Chinese remainder theorem* provides a formula to compute this solution:

$$x = \sum_{i=1}^{r} a_i M_i y_i \bmod M,$$

where $M_i = M/m_i$ and $y_i = {M_i}^{-1} \bmod m_i$, for $1 \leq i \leq r$. The following algorithm performs this computation.

Chinese Remainder Theorem (cont.)

Algorithm: *CRT*$(m_1, \ldots, m_r, a_1, \ldots, a_r)$

external *MultInv*

$M \leftarrow 1$

for $i \leftarrow 1$ **to** r

  do $M \leftarrow M \times m_i$

for $i \leftarrow 1$ **to** r

  do $\begin{cases} M_i = M/m_i \\ y_i = \textit{MultInv}(m_i, M_i) \end{cases}$

$x \leftarrow 0$

for $i \leftarrow 1$ **to** r

  do $\{x \leftarrow x + a_i \times M_i \times y_i \bmod M$

return (x)

Chinese Remainder Theorem (example)

Suppose $r = 3$, $m_1 = 7$, $m_2 = 11$ and $m_3 = 13$. Then $M = 1001$. We compute $M_1 = 143$, $M_2 = 91$ and $M_3 = 77$, and then $y_1 = 5$, $y_2 = 4$ and $y_3 = 12$. Then the solution to the given system

$$\begin{aligned} x &\equiv a_1 \pmod 7 \\ x &\equiv a_2 \pmod{11} \\ x &\equiv a_3 \pmod{13} \end{aligned}$$

is

$$x = (715a_1 + 364a_2 + 924a_3) \bmod 1001.$$

For example, if $a_1 = 5$, $a_2 = 3$ and $a_3 = 10$, then

$$\begin{aligned} x &= (715 \times 5 + 364 \times 3 + 924 \times 10) \bmod 1001 \\ &= 13907 \bmod 1001 \\ &= 894. \end{aligned}$$

Square-and-multiply Algorithm

We compute $z = x^c$ in any multiplicative group. The exponent, c, is represented in binary notation, say $c = \sum_{i=0}^{\ell-1} c_i 2^i$, where $c_i = 0$ or 1, $0 \leq i \leq \ell - 1$.

Algorithm: *Square-and-multiply*(x, c)

external *GroupMult*

$z \leftarrow$ **id**

for $i \leftarrow \ell - 1$ **downto** 0

do $\begin{cases} z \leftarrow \textit{GroupMult}(z, z) \\ \textbf{if } c_i = 1 \\ \quad \textbf{then } z \leftarrow \textit{GroupMult}(z, x) \end{cases}$

return (z)

Square-and-multiply Algorithm (example)

We compute $9726^{3533} \bmod 11413$. The exponent $c = 3533$ has binary representation $c = 110111001101_2$.

i	c_i	z	i	c_i	z
11	1	$1^2 \times 9726 = 9726$	10	1	$9726^2 \times 9726 = 2659$
9	0	$2659^2 = 5634$	8	1	$5634^2 \times 9726 = 9167$
7	1	$9167^2 \times 9726 = 4958$	6	1	$4958^2 \times 9726 = 7783$
5	0	$7783^2 = 6298$	4	0	$6298^2 = 4629$
3	1	$4629^2 \times 9726 = 10185$	2	1	$10185^2 \times 9726 = 105$
1	0	$105^2 = 11025$	0	1	$11025^2 \times 9726 = 5761$

Rabin Decryption

Suppose that $p \equiv q \equiv 3 \pmod 4$, where $n = pq$ is the public modulus. The input will be a ciphertext, $y \in \mathbb{Z}_n{}^*$. The output should be the four possible plaintexts, i.e., the elements $x \in \mathbb{Z}_n{}^*$ such that $x^2 \equiv y \pmod n$. It is apparent that $x^2 \equiv y \pmod n$ if and only if $x^2 \equiv y \pmod p$ and $x^2 \equiv y \pmod q$. Because p and q are both congruent to 3 modulo 4, we can compute the two square roots of y modulo p, and the two square roots of y modulo q, using the formula presented earlier. Then we can use the *Chinese remainder theorem* to compute the four square roots of y modulo n.

Rabin Decryption (cont.)

Algorithm: *RabinDecrypt*(n, p, q, y)
external *CRT*
if $(y^{(p-1)/2} \not\equiv 1 \pmod{p})$ **or** $(y^{(q-1)/2} \not\equiv 1 \pmod{q})$
then return ("y is not a valid ciphertext")

$$\text{else} \begin{cases} x_p \leftarrow y^{(p+1)/4} \bmod p \\ x_q \leftarrow y^{(q+1)/4} \bmod q \\ x_1 \leftarrow \mathit{CRT}(p, q, x_p, x_q) \\ x_2 \leftarrow \mathit{CRT}(p, q, x_p, q - x_q) \\ x_3 \leftarrow \mathit{CRT}(p, q, p - x_p, x_q) \\ x_4 \leftarrow \mathit{CRT}(p, q, p - x_p, q - x_q) \\ \textbf{return} \text{ ("the four plaintexts are } x_1, x_2, x_3 \text{ and } x_4\text{")} \end{cases}$$

Fundamental Computational Problems

A relatively small number of computational problems underly most public-key schemes. The most important of these are the **Integer Factoring** problem, and the **Discrete Logarithm** problem in a finite abelian group.

Problem

Integer Factoring
Instance: *A positive integer* $n \geq 2$.
Question: *Find the factorization of* n *into primes.*

In the context of cryptography, a composite integer n is commonly constructed as the product of two primes of roughly the same size. In order for this special case of the factoring problem to be intractable, it is currently recommended that n be a 1024-bit integer (or larger); see Lenstra and Verheul (2001).

Fundamental Computational Problems (cont.)

Problem

Discrete Logarithm

Instance: *A multiplicative group* $(G, \cdot)$*, an element* $\alpha \in G$ *having order* n*, and an element* $\beta \in \langle \alpha \rangle$*.*

Question: *Find the unique integer* a*,* $0 \leq a \leq n-1$*, such that*

$$\alpha^a = \beta.$$

We will denote this integer a *by* $\log_\alpha \beta$*.*

The **Discrete Logarithm** problem can be easy or difficult to solve, depending on the way that the group G is presented.

Settings for the Discrete Logarithm Problem

The most important settings (G, α) for the **Discrete Logarithm** problem in cryptographic applications are the following:

1. $G = (\mathbb{Z}_p{}^*, \cdot)$, p prime, α a primitive element modulo p, where $p \approx 2^{1024}$
2. $G = (\mathbb{Z}_p{}^*, \cdot)$, p, q prime, $p \equiv 1 \bmod q$, α an element in $\mathbb{Z}_p$ having order q, where $p \approx 2^{1024}$, $q \approx 2^{160}$
3. $G = (\mathbb{F}_{2^n}{}^*, \cdot)$, $n \approx 2^{1024}$ α a primitive element in $\mathbb{F}_{2^n}{}^*$
4. $G = (E, +)$, where E is an elliptic curve modulo a prime p, $\alpha \in E$ is a point having prime order $q = \#E/h$, where (typically) $h = 1, 2$ or 4, and $p, q \approx 2^{160}$.
5. $G = (E, +)$, where E is an elliptic curve over a finite field $\mathbb{F}_{2^n}$, $\alpha \in E$ is a point having prime order $q = \#E/h$, where (typically) $h = 2$ or 4, $q \approx 2^{160}$ and $n \approx 160$.

Table of Contents

What does the term "secure" mean?

In the August 1977 issue of **Scientific American**, Martin Gardner wrote a column on the newly developed RSA public-key cryptosystem entitled "A new kind of cipher that would take millions of years to break" Included in the article was a challenge ciphertext, encrypted using a 512-bit RSA key. The challenge was solved 17 years later, on April 26, 1994 by factoring the given public key (the plaintext was "The Magic Words are Squeamish Ossifrage").

What went wrong

The statement that the cipher would take millions of years to break probably referred to how long it would take to run the best factoring algorithm known in 1977 on the fastest computer available in 1977. However, between 1977 and 1994:

- computers became much faster
- improved factoring algorithms were found
- the development of the internet facilitated large-scale distributed computations

Even so, the factorization still required over 5000 MIPS-years of computation time in 1994.
The current state-of-the-art is the factorization of the 232-digit (768 bit) challenge, RSA-768, in December 2009.

A Formal Model

Any discussion of cryptographic security requires a specification of an attack model, an adversarial goal, and a level of security. These terms are defined as follows:

attack model

We will always assume that the adversary knows the protocol being used (this is called **Kerckhoff's Principle**) as well as the public key (if the system is a public-key system). We also assume that the adversary does not know any secret or private keys being used. Possible additional information provided to the adversary is specified in the attack model.

A Formal Model (cont.)

adversarial goal

The adversarial goal specifies what it means to "break" the cryptosystem. In other words, what is the adversary attempting to do and what information is he trying to determine? How is the notion of a "successful attack" defined?

level of security

The security level attempts to quantify the effort required to break the cryptosystem. Equivalently, what computational resources does the adversary have access to?

A statement of security for a cryptographic scheme will assert that a particular adversarial goal cannot be achieved in a specified attack model, given specified computational resources.

Attack Models for Cryptosystems

known ciphertext attack

The adversary is given a string of ciphertext (all encrypted with the same unknown key).

known plaintext attack

The adversary is given a string of plaintext and the corresponding ciphertext.

chosen plaintext attack

The adversary chooses a string of plaintext and is then given the corresponding ciphertext.

chosen ciphertext attack

The adversary chooses a string of ciphertext and is then given the corresponding plaintext

Adversarial Goals

complete break

The adversary determines the private (or secret) key.

decryption of previously unseen ciphertexts

The adversary can decrypt a previously unseen ciphertext with some specified non-zero probability.

partial information

The adversary can determine some specific type of partial information about the plaintext, given a previously unseen ciphertext, with some specified non-zero probability.

distinguishability

This means that the adversary can distinguish between encryptions of two given plaintexts (**semantic security** means that the adversary cannot do this).

Security Levels

computational security

This means that a specific algorithm to break the system is computationally infeasible.

provable security

This refers to a situation where breaking the cryptosystem can be reduced in a complexity-theoretic sense to solving some underlying (assumed difficult) mathematical problem, or a "simpler" cryptographic protocol.

unconditional security

This means that the cryptosystem cannot be broken, even with unlimited computational resources, because there is not enough information available to the adversary for him to be able to do this.

One-time Pad

Let $n \geq 1$ be an integer, and take $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^n$. For $K \in (\mathbb{Z}_2)^n$, define $e_K(x)$ to be the vector sum modulo 2 of K and x (or, equivalently, the exclusive-or of the two associated bitstrings). So, if $x = (x_1, \ldots, x_n)$ and $K = (K_1, \ldots, K_n)$, then

$$e_K(x) = (x_1 + K_1, \ldots, x_n + K_n) \bmod 2.$$

Decryption is identical to encryption. If $y = (y_1, \ldots, y_n)$, then

$$d_K(y) = (y_1 + K_1, \ldots, y_n + K_n) \bmod 2.$$

Unconditional Security of the One-time Pad

- the *one-time pad* is unconditionally semantically secure in a known-ciphertext attack if the key is used only for a single encryption
- the attack model is that the adversary is given a single piece of ciphertext, y, as well as two possible plaintexts, x_1 and x_2
- the adversary is supposed to determine whether y is an encryption of x_1 or x_2.
- either situation is possible under the given information: y is the encryption of x_i $(i = 1, 2)$ if and only if $K = y \oplus x_i$.
- there is no way for the adversary to achieve his goal, regardless of how much computer time he uses, so we achieve unconditional security against the specified attack

Security of the Rabin Cryptosystem

- the *Rabin cryptosystem* achieves provable one-way encryption against a chosen-plaintext attack, assuming that factoring is infeasible
- the proof of security uses a **reduction**
- we prove that, if a decryption algorithm *Rabin decrypt* exists, then there exists a randomized algorithm that factors the modulus, n, with probability at least $1/2$
- we assume that n is the product of two distinct primes p and q; and *Rabin decrypt* is an oracle that performs Rabin decryption, returning one of the four possible plaintexts corresponding to a given valid ciphertext

Security of the Rabin Cryptosystem

Algorithm: *Rabin oracle factoring*(n)
external *Rabin decrypt*
choose a random integer $r \in \mathbb{Z}_n{}^*$
$y \leftarrow r^2 \bmod n$
$x \leftarrow$ *Rabin decrypt*(y)
if $x \equiv \pm r \pmod{n}$
then return ("failure")
else $\begin{cases} p \leftarrow \gcd(x+r, n) \\ q \leftarrow n/p \\ \textbf{return } (\text{“}n = p \times q\text{”}) \end{cases}$

Analysis

- $x^2 \equiv r^2 \pmod{n}$, and hence $n \mid (x+r)(x-r)$
- there are four plaintexts whose encryption yields the ciphertext y; two of these are r and $-r \bmod n$
- the probability that $x \neq - \pm r \bmod n$ is $1/2$
- in this case, we have that

$$\begin{aligned} (x+r)(x-r) &\equiv 0 \pmod{pq} \\ x+r &\not\equiv 0 \pmod{pq} \\ x-r &\not\equiv 0 \pmod{pq} \end{aligned}$$

- exactly one of p and q divide $x+r$.
- $\gcd(x+r, n)$ is one of the two prime factors of n and $\gcd(x-r, n)$ yields the other

An Attack on RSA

- the *RSA cryptosystem* does not provide computationally secure one-way encryption in the chosen-ciphertext model
- the attack uses the **homomorphic property** of the RSA encryption function: let $y_1, y_2 \in \mathbb{Z}_n$, and suppose $y_3 \equiv y_1 y_2 \pmod{n}$; then $d_K(y_1)d_K(y_2) \equiv d_K(y_3) \pmod{n}$
- suppose the adversary is given a ciphertext, say y, and is asked to decrypt it
- The adversary chooses $y_1, y_2 \in \mathbb{Z}_n$ such that $y \neq y_1, y_2$ and $y \equiv y_1 y_2 \pmod{n}$
- then he requests the values $x_1 = d_K(y_1)$ and $x_2 = d_K(y_2)$
- finally, the adversary easily computes $x = x_1 x_2 \bmod n$, which is the decryption of y

What Do Proofs of Security Mean?

Whether a proof of security has any meaning in the "real world" depends on several factors:

1. Does the attack model specify **all** the information available to the adversary, and his computational resources? If the adversary has additional information, then all bets are off.
2. Can the adversary do "bad things" even if he cannot achieve the specified adversarial goal?
3. How realistic are the underlying assumptions in the attack model? (This includes explicit and "hidden" assumptions.)

Clearly, in order for a security proof to be significant/valid/realistic, we should try to prove that a **weak** adversarial goal cannot be attained in a **strong** attack model, and we should minimize the assumptions required, to the greatest extent possible.

Assumptions

black-box model

Cryptographic schemes are most often modelled as "black boxes". Their input-output behaviour is described precisely, but details of their implementation (in software or hardware) are not taken into consideration in security proofs in this model. A security proof in the black box model may not be valid if details of the implementation are known to the adversary. Side channel attacks provide illustrations of this.

intractibility assumptions

The presumed difficulty of certain computational problems are often used in "provable security" results for public-key schemes. Commonly used intractibility assumptions are based on the apparent difficulty of factoring large integers, or of solving the discrete logarithm problem in certain large abelian groups.

Assumptions (cont.)

random oracle model

Many cryptographic schemes use hash functions (in practice, *SHA-1* is the most common). Many security proofs assume that the ourputs of a hash function are truly random and unpredictable; this is the so-called **random oracle model**. A proof in the random oracle model presumes that the adversary cannot use any information about the actual hash function being used in the scheme in his attack. Stated another way, a security proof in the random oracle model shows that the adversary cannot achieve his goal without exploiting weaknesses of the hash function used in the scheme.

Assumptions (cont.)

random number generators

Many randomized cryptographic schemes require the generation of random numbers. Most security proofs of such schemes assume implicitly that there is some ideal random number generator that outputs truly random numbers for use in the scheme. It would be better and more accurate to include this as an explicit assumption in the attack model. In analogy with the random oracle model, we might call this the **perfect random number generator model**. In practice, any scheme requiring random numbers will make use of a certain pseudorandom number generator. A security proof in the perfect random number generator model would show that the adversary cannot achieve his goal without exploiting weaknesses of the particular pseudorandom number generator used in the scheme.

Scenarios for Provable Security

1. Proving that cryptographic schemes are secure, provided that an underlying computaitonal problem is intractable.
2. Proving that a secure basic scheme can be used to build a secure general scheme with increased functionality.
3. Proving that a basic scheme providing a minimal level of security can be used to build a scheme with a higher level of security.
4. Proving that a cryptographic protocol is secure if the underlying cryptographic tools used in the protocol are secure.

Attack Models for Signature Schemes

key-only attack

The adversary possesses Alice's public key, i.e., the verification function, ver_K.

known message attack

The adversary possesses a list of messages previously signed by Alice, say $(x_1, y_1), (x_2, y_2), \ldots,$ where the x_i's are messages and the y_i's are Alice's signatures on these messages (so $y_i = sig_K(x_i)$, $i = 1, 2, \ldots$).

chosen message attack

The adversary requests Alice's signatures on a list of messages. Therefore he chooses messages $x_1, x_2, \ldots,$ and Alice supplies her signatures on these messages, namely, $y_i = sig_K(x_i)$, $i = 1, 2, \ldots$.

Adversarial Goals for Signature Schemes

total break

The adversary is able to determine Alice's private key, i.e., the signing function sig_K.

selective forgery

With some non-negligible probability, the adversary is able to create a valid signature on a new (random) message chosen by someone else. In other words, if the adversary is given a message x, then he can determine (with some probability) a signature y such that $ver_K(x, y) = \text{true}$.

existential forgery

The adversary is able to create a valid signature for at least one new message. In other words, the adversary can create a pair (x, y) where x is a message and $ver_K(x, y) = \text{true}$.

An Attack on the RSA Signature Scheme

Suppose the *RSA Signature Scheme* is used without a hash function. It is a simple matter for an adversary to choose an arbitrary $y \in \mathbb{Z}_n$ and then compute $x = y^b \bmod n$. Clearly y is a valid signature on the message x, so this is a valid forgery. The attack is an existential forgery in the key-only model.

Note that this attack is prevented by using a preimage resistant hash function.

Table of Contents

Identification Techniques: What you are, what you have, what you know

physical attributes

People often identify other people already known to them by their appearance. This could include family and friends as well as famous celebrities. Attributes that are unique to an individual include fingerprints or retina scans (**biometrics**).

credentials

Trusted documents or cards such as driver's licences and passports function as credentials in many situations. Credentials often include pictures, which enables physical identification, as described above.

Identification Techniques (cont.)

knowledge

Knowledge is often used for identification when the person being identified is not in the same physical location as the the person or entity performing the identification. In the context of identification, knowledge could be a password or PIN (personal identification number), or "your mother's maiden name" (a favorite of credit card companies). The difficulty with using knowledge for identification is that such knowledge may not be secret in the first place, and, moreover, it is usually revealed as part of the identification process. This allows for possible future impersonation of the person being identified, which is not a good thing!

Identification Scenarios

telephone "calling cards"

require the knowledge of the telephone number that is being billed for the call, together with a four-digit PIN

remote login

requires a valid user name and the corresponding password.

non-chip credit card purchases

usually require verification that the customer's signature matches the signature on the back of the card.

credit card purchases without the credit card

require a valid credit card number, the expiry date and the CVV code.

bank machine withdrawals

require a bank card together with a four-digit PIN.

Attack Model and Adversarial Goal

- suppose Alice is identifying herself to Bob
- the attack model is that the adversary can observe all the information being transmitted between Alice and Bob
- the adversarial goal will be to **impersonate** Alice (and perhaps Bob is the adversary!)
- a secure scheme requires randomization, typically a "random challenge" by Bob; otherwise the adversary can just **replay** a previous session

Interactive Protocols

- an **interactive protocol** will comprise two (or more) parties that are communicating with each other
- each party is modelled as an algorithm that alternately sends and receives information
- each **session** consists of two or more **flows** (a message sent from Alice to Bob counts as one flow)
- at the end of a session, Bob "accepts" or "rejects" (this is Bob's **internal state** at the end of the protocol)

Protocols for Identification

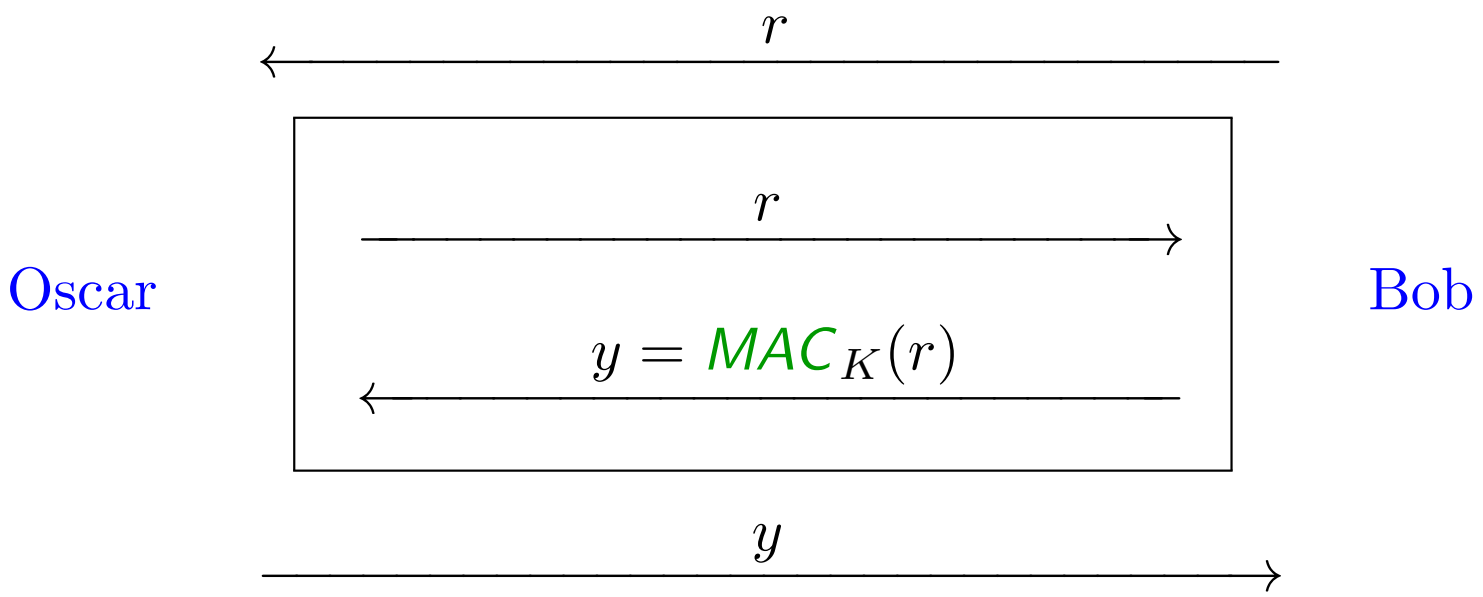
- the cryptographic tool used in an identification scheme is either a **signature scheme** (in the public-key setting) or a **message authentication code** (in the secret-key setting)
- we first consider the secret-key setting, assuming that Alice and Bob both hold the same secret key, K
- a message authentication code MAC_K will used to compute authentication tags for messages and/or challenges sent from Alice to Bob or vice versa

Protocol: *Insecure Challenge-and-Response Scheme*

1. Bob chooses a random **challenge**, r, which he sends to Alice.
2. Alice computes $y = MAC_K(r)$ and sends y to Bob.
3. Bob computes $y' = MAC_K(r)$. If $y' = y$, then Bob outputs "accept"; otherwise, Bob outputs "reject".

Alice		Bob
	$\xleftarrow{\quad r \quad}$	
$y = MAC_K(r)$		
	$\xrightarrow{\quad y \quad}$	
		$y = MAC_K(r)$?

A Parallel Session Attack



In this attack, Oscar pretends to be Alice. After receiving a challenge from Bob, Oscar initiates a second "parallel" session with Bob, in which he asks Bob to identify himself using the same challenge, r. Oscar can send Bob's tag y (from the inner session) back to Bob in the outer session.

A Secure Variation?

The following protocol prevents the previous attack:

Protocol: *Secure Challenge-and-Response Scheme*

1. Bob chooses a random challenge, r, which he sends to Alice.
2. Alice computes
$$y = MAC_K(Alice \parallel r)$$
and sends y to Bob.
3. Bob computes
$$y' = MAC_K(Alice \parallel r).$$
If $y' = y$, then Bob "accepts"; otherwise, Bob "rejects".

Note: identifiers (such as "Alice") and random challenges should be strings of a fixed, predetermined length, e.g., 100 bits each.

Assumptions Required for a Security Proof

1. The secret key, K is known only to Alice and Bob.
2. Alice and Bob both have perfect random number generators which they use to determine their challenges. Therefore, the probability that the same challenge occurs by chance in two different sessions is very small. If r is a 100-bit string, then each possible challenge occurs with probability 2^{-100} in a given session.
3. The message authentication code is "secure". In other words, the probability that Oscar can correctly compute $MAC_K(x)$ is very small, even when he is given many other valid tags, say $MAC_K(x_i)$, $i = 1, 2, \ldots, 1000000$, provided that $x \neq x_i$ for any i.

Security Proof (informal)

1. Could the value $y = MAC_K(Alice \parallel r)$ have previously been transmitted by Bob himself in some other session?
   No: Bob only computes tags of the form $MAC_K(Bob \parallel r)$, so he would not have created y himself.
2. Suppose the value y was previously constructed by Alice in some other session. The challenge r is assumed to be a challenge newly created by Bob, so it is highly probable that Bob would **not** have issued the same challenge in some other session.
3. Suppose that y is a "new" tag that is constructed by Oscar (i.e., it's not copied from a previous session). This contradicts the assumed security of the message authentication code.

MAC Security

- if we can prove an explicit, precise statement of the security of the underlying MAC, then we can give a precise security guarantee for the identification protocol
- alternatively, if we make an assumption about the MAC's security, then we can provide a security result for the identification protocol that depends on this assumption (thus we achieve **provable security**)
- a MAC is said to be (T, q, ϵ)-**secure** if the adversary cannot construct a valid tag for any new message with probability greater than ϵ in time T, given that the adversary has previously observed valid tags for at most q messages (this would refer to a known-message attack)
- suppose that the random challenge is a k-bit string $r \in \{0, 1\}^k$

Analysis

1. The tag $y = MAC_K(Alice \parallel r)$ would not have been previously constructed by Bob in some other session.
2. Suppose the value y was previously computed by Alice in some other session. The probability that Bob already used r in a specific previous session is at most $1/2^k$. There are at most q previous sessions under consideration, so the probability that r was used as challenge in one of these previous sessions is at most $q/2^k$.
3. Suppose the tag y is a "new" tag that is constructed by Oscar. Then, Oscar will succeed with probability at most ϵ, given that he has computing time T; this follows from the security of the message authentication code being used.

Summing up, Oscar's probability of deceiving Bob in time T is at most $q/2^k + \epsilon$.

An Attack?

Consider the following **intruder-in-the-middle** scenario:

Alice	Oscar	Bob
$\xleftarrow{\quad r \quad}$		$\xleftarrow{\quad r \quad}$
$\xrightarrow{y = MAC_K(Alice \parallel r)}$		$\xrightarrow{\quad y \quad}$

- At the end of the session, Bob will accept
- Should we consider this to be a successful attack by Oscar?

Analysis

- We do **not** consider the previous scenario to be a real attack, because Alice has successfully identified herself to Bob in this session.
- Bob issued a challenge r and Alice computed the correct response y to the challenge.
- Oscar simply forwarded messages to their intended recipients without modifying the messages, so Oscar was not an active participant in the protocol.
- The session executed exactly as it would have if Oscar had not been present.

Adversarial Goals, Revisited

- We formulate the notion of adversarial goal more precisely.
- We will say that the adversary (Oscar) is **active** in a particular session if one of the following conditions holds:
  1. Oscar creates a new message and places it in the channel
  2. Oscar changes a message in the channel, or
  3. Oscar diverts a message in the channel so it is sent to someone other than the intended receiver
- According to this definition, Oscar is not active in the intruder-in-the-middle scenario considered above
- The goal of the adversary is to have the initiator of the protocol (e.g., Bob) "accept" in some session **in which the adversary is active**.
- Note that **disruption** is not considered a successful attack.

Assumptions regarding Honest Participants

- A participant in a session of the protocol (Alice or Bob) is said to be an **honest participant** if she/he follows the protocol, performs correct computations, and does not reveal any secret information to the adversary (namely, Oscar).
- If a participant is not honest, then it is very difficult to say anything meaningful about the security of a protocol.
- Statements of security require that participants are honest.

Attack Models

- Before he actually tries to deceive Bob, say, Oscar carries out an **information-gathering** phase
- Oscar is said to be **passive** during this phase if he simply observes sessions between Alice and Bob
- alternatively, we might consider an attack model in which Oscar is **active** during the information-gathering phase
- for example, Oscar might be given temporary access to an oracle that computes authentication tags $MAC_K(\cdot)$ for the key K being used by Alice and Bob
- during this time period, Oscar can successfully deceive Alice and Bob
- after the information-gathering phase, the MAC oracle is confiscated (and then Oscar carries out his attack)

Security

- The security analysis that we performed applies to both of these attack models
- The identification protocol is secure (the adversary's success probability is at most $q/2^k + \epsilon$ in time T) in the **passive** information-gathering model provided that the MAC is (q, ϵ, T)-secure against a **known message** attack
- The identification protocol is secure in the **active** information-gathering model provided that the MAC is (q, ϵ, T)-secure against a **chosen message** attack

Notions of Mutual Identification

- Suppose we have a protocol in which Alice and Bob are proving their identities to each other simultaneously
- This is called **mutual identification**
- Both participants are required to "accept" if a session of the protocol is to be considered a successfully completed session.
- The adversary could be trying to fool Alice, Bob, or both of them into accepting.
- The adversarial goal is to cause an honest participant to "accept" **after** a flow in which the adversary is active.

Secure Mutual Identification

- The following conditions specify what the outcome of a mutual identification protocol should be, if the protocol is secure.
  1. If Alice and Bob are honest and the adversary is passive, then Alice and Bob will both output "accept" in any session in which they are the two participants.
  2. If the adversary is active during a given flow of the protocol, then no honest participant will "accept" after that flow.
- Note that the adversary might be inactive in a particular session until after one participant accepts, and then become active. Therefore it is possible that one honest participant "accepts " and then the other honest participant "rejects". The adversary does not achieve his goal in this case, even though the session did not successfully complete.

Mutual Identification Protocol

The following protocol for mutual identification basically consists of two runs of the secure one-way protocol, modified so that the number of flows is reduced from four to three:

Protocol: *Insecure Mutual Challenge-and-Response Scheme*

1. Bob chooses a random challenge, r_1, which he sends to Alice.
2. Alice chooses a random challenge, r_2. She also computes $y_1 = MAC_K(Alice \parallel r_1)$ and sends r_2 and y_1 to Bob.
3. Bob computes $y_1' = MAC_K(Alice \parallel r_1)$. If $y_1' = y_1$, then Bob "accepts"; otherwise, Bob "rejects". Bob also computes $y_2 = MAC_K(Bob \parallel r_2)$ and sends y_2 to Alice.
4. Alice computes $y_2' = MAC_K(Bob \parallel r_2)$. If $y_2' = y_2$, then Alice "accepts"; otherwise, Alice "rejects".

A Parallel Session Attack

The protocol is insecure; Oscar can fool Alice in a parallel session attack:

Alice	Oscar	Bob
$\xleftarrow{\quad r_1 \quad}$		
$\xrightarrow{MAC_K(Alice \parallel r_1), r_2}$		
	$\xrightarrow{\quad r_2 \quad}$	
	$\xleftarrow{MAC_K(Bob \parallel r_2), r_3}$	
$\xleftarrow{MAC_K(Bob \parallel r_2)}$		

Secure Mutual Identification Protocol

The following variation is secure:

Protocol: *Secure Mutual Challenge-and-Response Scheme*

1. Bob chooses a random challenge, r_1, which he sends to Alice.
2. Alice chooses a random challenge, r_2. She also computes $y_1 = MAC_K(Alice \parallel r_1 \parallel r_2)$ and sends r_2 and y_1 to Bob.
3. Bob computes $y_1' = MAC_K(Alice \parallel r_1 \parallel r_2)$. If $y_1' = y_1$, then Bob outputs "accept"; otherwise, Bob outputs "reject". Bob also computes $y_2 = MAC_K(Bob \parallel r_2)$ and sends y_2 to Alice.
4. Alice computes $y_2' = MAC_K(Bob \parallel r_2)$. If $y_2' = y_2$, then Alice outputs "accept"; otherwise, Alice outputs "reject".

Secure Mutual Identification Protocol

Alice		Bob
	$\xleftarrow{r_1}$	
$y_1 = MAC_K(Alice \parallel r_1 \parallel r_2)$		
	$\xrightarrow{r_2, y_1}$	
		$y_1 = MAC_K(Alice \parallel r_1 \parallel r_2)?$ $y_2 = MAC_K(Bob \parallel r_2)$
	$\xleftarrow{y_2}$	
$y_2 = MAC_K(Bob \parallel r_2)?$		

Informal Analysis

- Note that any tag y_1 is computed "differently" than any tag y_2. Therefore it is impossible that a y_1 tag from one session can be re-used as a y_2 tag from another session (or vice versa), as was done in the parallel session attack.
- Let's try to prove informally that the protocol is secure against any attack.
- Oscar could try to play the role of Bob (fooling Alice) or Alice (fooling Bob), by determining y_2 or y_1, respectively.

Informal Analysis (cont.)

- Suppose that Oscar has seen at most q tags from previous sessions (this limits the number of previous sessions to $q/2$, because there are two tags sent per session).
- The probability that a tag y_1 or y_2 can be reused from a previous session is at most $q/2^k$.
- The probability that Oscar can compute a new y_1 tag is at most ϵ, and the probability that he can compute a new y_2 tag is at most ϵ.
- Therefore Oscar's probability of deceiving Alice or Bob is at most $q/2^k + 2\epsilon$.

The Public-key Setting

- assume that Alice and Bob are members of a network, in which every participant has public and private keys for certain prespecified cryptosystems and/or signature schemes
- it is necessary to provide a mechanism to authenticate public keys of other people in the network, i.e., a **public key infrastructure**
- we assume there is a **certification authority**, denoted CA, who signs the public keys of all people in the network
- the (public) verification key of the CA, denoted ver_{CA}, is assumed to be known "by magic" to everyone in the network
- a **certificate** for someone in the network will consist of some identifying information for that person, their public key(s), and the signature of the CA on that information

Certificates

Protocol: *Issuing a Certificate to Alice*

1. The CA establishes Alice's identity by means of conventional forms of identification such as a birth certificate, passport, etc. Then the CA forms a fixed length string $ID(Alice)$ which contains her identification information.
2. A private signing key for Alice, sig_{Alice}, and a corresponding public verification key, ver_{Alice}, are determined.
3. The CA generates its signature

$$s = sig_{CA}(ID(Alice) \parallel ver_{Alice})$$

on Alice's identity string and verification key. The certificate

$$\mathbf{Cert}(Alice) = (ID(Alice) \parallel ver_{Alice} \parallel s)$$

is given to Alice.

Verifying a Certificate

- Anyone who knows the CA's verification key, ver_{CA}, can verify anyone else's certificate.
- Suppose that Bob wants to be assured that Alice's public key is authentic.
- Alice can give her certificate to Bob. Bob can then verify the signature of the CA by checking that

$$ver_{CA}(ID(Alice) \parallel ver_{Alice}, s) = \textbf{true}.$$

- Certificates contain only public information, so they can be distributed to anyone.
- The security of a certificate follows immediately from the security of the signature scheme used by the CA.
- Certificates are verified before the identification protocol is executed.

Public-key Mutual Identification

- we want to modify protocols from the secret-key setting, replacing message authentication codes by signature schemes
- in the secret-key setting, we included the name of the person who produced the tag in the computation of the tag
- in the public-key setting, only one person can create signatures using a specified private signing key, so we do not need to explicitly designate who **created** a particular signature
- at the beginning of the protocol, each participant has an **intended peer** (i.e., the person they think they are communicating with)
- they will use the intended peer's verification key to verify all signatures received in the protocol

Protocol: *Public-key Mutual Identification Protocol*

1. Bob chooses a random challenge, r_1, which he sends to Alice.
2. Alice chooses a random challenge, r_2. She also computes $y_1 = \mathit{sig}_{Alice}(Bob \parallel r_1 \parallel r_2)$ and sends $\mathbf{Cert}(Alice)$, r_2 and y_1 to Bob.
3. Bob verifies Alice's public key, ver_{Alice}, on the certificate $\mathbf{Cert}(Alice)$. Then he checks that $\mathit{ver}_{Alice}(Bob \parallel r_1 \parallel r_2, y_1) = \mathbf{true}$. If so, then Bob "accepts"; otherwise, Bob "rejects". Bob also computes $y_2 = \mathit{sig}_{Bob}(Alice \parallel r_2)$ and sends $\mathbf{Cert}(Bob)$ and y_2 to Alice.
4. Alice verifies Bob's public key, ver_{Bob}, on the certificate $\mathbf{Cert}(Bob)$. Then she checks that $\mathit{ver}_{Bob}(Alice \parallel r_2, y_2) = \mathbf{true}$. If so, then Alice "accepts"; otherwise, Alice "rejects".

Public-key Mutual Identification Protocol

Alice		Bob
	$\xleftarrow{r_1}$	
$y_1 = \mathit{sig}_{Alice}(Bob \parallel r_1 \parallel r_2)$		
	$\xrightarrow{r_2, y_1}$	
		$\mathit{ver}_{Alice}(Bob \parallel r_1 \parallel r_2, y_1) = \textbf{true}?$ $y_2 = \mathit{sig}_{Bob}(Alice \parallel r_2)$
	$\xleftarrow{y_2}$	
$\mathit{ver}_{Bob}(Alice \parallel r_2, y_2) = \textbf{true}?$		

Security of the Protocol

- the preceding protocol is provably secure
- it is secure if the signature scheme is secure and challenges are generated randomly
- interestingly, a slight modification of the scheme is insecure
- the insecure modified protocol includes a third random number r_3 that is signed by Bob
- the protocol becomes vulnerable to a parallel session attack (exercise for the reader)

Protocol: *Insecure Public-key Mutual Identification*

1. Bob chooses a random challenge, r_1, which he sends to Alice.
2. Alice chooses a random challenge, r_2. She also computes $y_1 = \mathit{sig}_{Alice}(Bob \parallel r_1 \parallel r_2)$ and sends $\mathbf{Cert}(Alice)$, r_2 and y_1 to Bob.
3. Bob verifies Alice's public key, ver_{Alice}, on the certificate $\mathbf{Cert}(Alice)$. Then he checks that $\mathit{ver}_{Alice}(Bob \parallel r_1 \parallel r_2, y_1) = \mathbf{true}$. If so, then Bob "accepts"; otherwise, Bob "rejects". Bob also chooses a random number r_3, computes $y_2 = \mathit{sig}_{Bob}(Alice \parallel r_2 \parallel r_3)$ and sends $\mathbf{Cert}(Bob)$, r_3 and y_2 to Alice.
4. Alice verifies Bob's public key, ver_{Bob}, on the certificate $\mathbf{Cert}(Bob)$. Then she checks that $\mathit{ver}_{Bob}(Alice \parallel r_2 \parallel r_3, y_2) = \mathbf{true}$. If so, then Alice "accepts"; otherwise, Alice "rejects".

Insecure Public-key Mutual Identification Protocol

Alice		Bob
	$\xleftarrow{r_1}$	
$y_1 = \mathit{sig}_{Alice}(Bob \parallel r_1 \parallel r_2)$		
	$\xrightarrow{r_2, y_1}$	
		$\mathit{ver}_{Alice}(Bob \parallel r_1 \parallel r_2, y_1) = \textbf{true}?$ $y_2 = \mathit{sig}_{Bob}(Alice \parallel r_2 \parallel r_3)$
	$\xleftarrow{r_3, y_2}$	
$\mathit{ver}_{Bob}(Alice \parallel r_2 \parallel r_3, y_2) = \textbf{true}?$		

Specialized Identification Schemes

- Another approach used to construct identification schemes is to design schemes "from scratch", without using any other cryptographic tools as building blocks.
- A potential advantage of schemes of this type is that they might be more efficient and have a lower communication complexity than schemes considered previously
- Such schemes can be based on the discrete logarithm problem:

Problem

Discrete Logarithm

Instance: *A multiplicative group* $(G, \cdot)$*, an element* $\alpha \in G$ *having order* n*, and an element* $\beta \in \langle \alpha \rangle$*.*

Question: *Find the unique integer* a*,* $0 \leq a \leq n-1$*, such that* $\alpha^a = \beta$*.*

The Schnorr Scheme (setup)

- We take α to be an element having prime order q in the group ${\mathbb{Z}_p}^*$ (where p is prime and $p - 1 \equiv 0 \pmod{q}$).
- Such an α can be obtained by raising a primitive element in ${\mathbb{Z}_p}^*$ to the $(p-1)/q$th power.
- This setting of the **Discrete Logarithm Problem** is thought to be secure if $p \approx 2^{1024}$ and $q \approx 2^{160}$
- Values of p, q and α satisfying these conditions are chosen by a CA and made public to all users of a network.
- These values could be included on Alice's and/or the CA's certificate, for example.

The Schnorr Scheme (setup, cont.)

- Alice chooses a **private key** a, where $0 \leq a \leq q-1$, and constructs a **public key** $v = \alpha^{-a} \bmod p$.
- v can be computed as $(\alpha^a)^{-1} \bmod p$, or (more efficiently) as $\alpha^{q-a} \bmod p$.
- The CA signs Alice's public key and puts it on Alice's certificate.
- The value $t = 40$ is a public **security parameter** (the adversary's probability of deceiving Alice or Bob will be 2^{-t}, so $t = 40$ is sufficient for most applications
- The *Schnorr Identification Scheme* is a three-flow protocol in which Alice identifies herself to Bob, say.

Protocol: *Schnorr Identification Scheme*

1. Alice chooses a random number, k, where $0 \leq k \leq q-1$ and computes a **commitment** $\gamma = \alpha^k \bmod p$. She sends $\mathbf{Cert}(Alice)$ and γ to Bob.
2. Bob verifies Alice's public key, v, on the certificate $\mathbf{Cert}(Alice)$. Bob chooses a random **challenge** r, $1 \leq r \leq 2^t$ and sends r to Alice.
3. Alice computes $y = k + ar \bmod q$ and sends the **response** y to Bob.
4. Bob verifies that $\gamma \equiv \alpha^y v^r \pmod p$. If so, then Bob "accepts"; otherwise, Bob "rejects".

Completeness

- If Alice and Bob are honest, the protocol will run successfully:

$$\begin{aligned} \alpha^y v^r &\equiv \alpha^{k+ar}\alpha^{-ar} \pmod{p} \\ &\equiv \alpha^k \pmod{p} \\ &\equiv \gamma \pmod{p}. \end{aligned}$$

- This is is called the **completeness** property of the protocol.

An Example

Suppose $p = 88667$, $q = 1031$ and $t = 10$. The element $\alpha = 70322$ has order q in $\mathbb{Z}_p{}^*$. Suppose Alice's private key is $a = 755$; then

$$\begin{aligned} v &= \alpha^{-a} \bmod p \\ &= 70322^{1031-755} \bmod 88667 \\ &= 13136. \end{aligned}$$

Now suppose Alice chooses $k = 543$. Then she computes

$$\begin{aligned} \gamma &= \alpha^{k} \bmod p \\ &= 70322^{543} \bmod 88667 \\ &= 84109. \end{aligned}$$

and sends γ to Bob.

An Example (cont.)

Suppose Bob issues the challenge $r = 1000$. Then Alice computes

$$\begin{aligned} y &= k + ar \bmod q \\ &= 543 + 755 \times 1000 \bmod 1031 \\ &= 851 \end{aligned}$$

and sends her response, y, to Bob. Bob then verifies that

$$84109 \equiv 70322^{851} 13136^{1000} \pmod{88667}.$$

Efficiency of the Protocol

Alice		Bob
$k \in_R \{0, \ldots, q-1\}$ $\gamma = \alpha^k \bmod p$	$\xrightarrow{\gamma \quad (1024 \text{ bits})}$	
	$\xleftarrow{r \quad (40 \text{ bits})}$	$r \in_R \{1, \ldots, 2^t\}$
$y = k + ar \bmod q$	$\xrightarrow{y \quad (160 \text{ bits})}$	$\gamma \equiv \alpha^y v^r \pmod p$?

- Alice's computations could be done by a smart card; in particular, γ can be precomputed
- the number of bits communicated is small
- γ can be replaced by *SHA-1*(γ) to provide additional efficiency (note that the scheme would have to be modified slightly if this change is made)

Proof of Knowledge: Security wrt Impersonations

- Can Oscar impersonate Alice?
- Oscar can guess Bob's challenge r with probability $1/2^t$ and precompute γ and y that work for the given r
- Namely, Oscar can choose y at random and compute

$$\gamma = \alpha^y v^r \bmod p$$

- On the other hand, suppose for some γ that Oscar somehow knows two values r_1 and r_2 such that he can compute correct responses y_1 and y_2, respectively
- Then

$$\gamma \equiv \alpha^{y_1} v^{r_1} \equiv \alpha^{y_2} v^{r_2} \pmod p,$$

so

$$\alpha^{y_1 - y_2} \equiv v^{r_2 - r_1} \pmod p.$$

Proof of Knowledge: Security wrt Impersonations (cont.)

- Since $v = \alpha^{-a}$, we have that

$$\alpha^{y_1 - y_2} \equiv \alpha^{a(r_1 - r_2)} \pmod{p},$$

and hence

$$y_1 - y_2 \equiv a(r_1 - r_2) \pmod{q}.$$

- $0 < |r_2 - r_1| < 2^t$ and $q > 2^t$ is prime.
- Hence, Oscar can compute

$$a = (y_1 - y_2)(r_1 - r_2)^{-1} \bmod q.$$

- Therefore Oscar can easily compute the value of a, which we are assuming is infeasible to compute in the given group.

Proof of Knowledge: Mathematical Analysis

- It is fair to ask under what conditions Oscar can compute some γ such that he knows two values r_1 and r_2 for which he can compute correct responses y_1 and y_2, respectively.
- Oscar's ability to do this will certainly depend on how successful he is in being able to impersonate Alice.
- We will prove the following theorem due to Schnorr:

Theorem

Suppose that A is an algorithm that succeeds in completing the Schnorr Identification Scheme (impersonating Alice) with success probability $\epsilon \geq 2^{-t+1}$. Then A can be used as an oracle in an algorithm B that computes Alice's private key, where B is a Las Vegas algorithm having expected complexity $O(1/\epsilon)$.

Mathematical Analysis (cont.)

We will use the following lemma a bit later.

Lemma

For $\delta \approx 0$, it holds that $(1-\delta)^{c/\delta} \approx e^{-c}$.

- Define a $q \times 2^t$ matrix $M = (m_{k,r})$, having entries equal to 0 or 1, where

$$m_{k,r} = 1 \text{ if and only if } A(k,r) \text{ executes successfully}$$

  (recall that $\gamma = \alpha^k$).
- Since A has success probability ϵ, it follows that M contains exactly $\epsilon q 2^t$ 1's.
- The **average** number of 1's in a row k of M is $\epsilon 2^t$.

Mathematical Analysis (cont.)

- Define a row k of M to be a **heavy row** if it contains more than $\epsilon 2^{t-1}$ 1's.
- Observe that a heavy row contains at least two 1's, because $\epsilon \geq 2^{-t+1}$.

Lemma

Given a random $m_{k,r} = 1$, the probability that k is a heavy row is at least $1/2$.

Proof.

The total number of 1's in light rows is at most $q \times \epsilon 2^{t-1} = \epsilon q 2^{t-1}$. Hence, the total number of 1's in heavy rows is at least $\epsilon q 2^t - \epsilon q 2^{t-1} = \epsilon q 2^{t-1}$. □

Mathematical Analysis (cont.)

We can now describe the algorithm B.

1. Choose random pairs (k, r) and run $A(k, r)$, until $M(k, r) = 1$. Define (k_1, r_1) to be the current pair (k, r) and proceed to step 2.
2. Denote by u the number of trials that were required in step 1. Choose random pairs (k_1, r) and run $A(k_1, r)$, until $M(k_1, r) = 1$ or until $4u$ trials have finished unsuccessfully.
3. If step 2 terminates with a successful pair (k_1, r_2) where $r_2 \neq r_1$, then we already showed that it is easy to compute Alice's private key and the algorithm B terminates successfully. Otherwise, go back to step 1 and start again.

Mathematical Analysis (cont.)

Since ϵ is the success probability of A, it follows that $E[u] = 1/\epsilon$, where $E[u]$ denotes the expected number of trials in step 1.

Lemma

$\Pr[u > 1/(2\epsilon)] \approx 0.6$.

Proof.

$u > 1/(2\epsilon)$ if and only if the first $1/(2\epsilon)$ random trials are unsuccessful. This happens with probability $(1-\epsilon)^{1/(2\epsilon)} \approx e^{-1/2} \approx 0.6$ (applying the Lemma stated previously). □

Mathematical Analysis (cont.)

Lemma

Suppose that $u > 1/(2\epsilon)$ and suppose also that k_1 is a heavy row. Then the probability that step 2 of B succeeds in finding a value r_2 such that $A(k_1, r_2) = 1$ is at least .37.

Proof.

Under the given assumptions, $4u > 2/\epsilon$. The probability that a random entry in row k_1 is a 1 is at least $\epsilon/2$. Therefore the probability that step 2 of B succeeds is at least

$$1 - \left(1 - \frac{\epsilon}{2}\right)^{2/\epsilon} \approx 1 - e^{-1} \approx 0.37.$$

□

Mathematical Analysis (cont.)

Lemma

Given that k_1 is a heavy row and step 2 of B succeeds in finding a pair (k_1, r_2) such that $A(k_1, r_2) = 1$, the probability that $r_2 \neq r_1$ is at least $1/2$.

Proof.

This follows immediately from the fact that a heavy row contains at least two 1's. □

Completing the Proof

Now we can analyze the expected complexity of algorithm B. We have shown that:

1. The probability that step 1 terminates with k being a heavy row is at least $1/2$.
2. The probability that $u > 1/(2\epsilon)$ is 0.6.
3. Assuming 1 and 2 hold, the probability that step 2 of B succeeds is at least .37.
4. Given 1, 2 and 3 all hold, the probability that $r_2 \neq r_1$ in step 3 of B is at least $1/2$.

The probability that 1–4 all hold (and hence algorithm B succeeds) is at least

$$\frac{1}{2} \times 0.6 \times 0.37 \times \frac{1}{2} \approx 0.055.$$

Therefore, the **expected number of iterations** of steps 1 and 2 in B is at most $1/0.055 \approx 18$.

Completing the Proof (cont.)

Finally, we determine the complexity of executing the operations in **one iteration** of steps 1 and 2 of B.

- The expected complexity of step 1 of B (i.e., the expected number of trials in step 1) is $E[u] = 1/\epsilon$.
- The complexity of step 2 is at most $4u$.
- Therefore, the expected complexity of steps 1 and 2 of B is at most $5/\epsilon$.

Therefore, the expected complexity of B is at most

$$18 \times \frac{5}{\epsilon} = \frac{90}{\epsilon} \in O\left(\frac{1}{\epsilon}\right).$$

Zero-knowledge Proofs of Knowledge

- The protocol is known as a **proof of knowledge** because Alice (the **prover**) cannot carry out the protocol successfully unless she "knows" the secret discrete logarithm, a.
- A **zero-knowledge** proof of knowledge is one in which no information about a is revealed to Bob (the **verifier**) by the protocol.
- We will show that the *Schnorr scheme* is a zero-knowledge proof of knowledge for an **honest verifier** (i.e., for a verifier who chooses his challenges r at random, as specified by the protocol).
- This provides fairly convincing evidence of security of the scheme.
- We require the notion of a **transcript** of a session, which consists of a triple $T = (\gamma, r, y)$ in which $\gamma \equiv \alpha^y v^r \pmod{p}$.

Probability Distributions of Transcripts

- The verifier can obtain a transcript $T(S)$ for each session S.
- The set of possible transcripts is

$$\mathcal{T} = \{(\gamma, r, y) : 1 \leq r \leq 2^t, 0 \leq y \leq q-1, \gamma \equiv \alpha^y v^r \pmod{p}\}.$$

- $|\mathcal{T}| = q\,2^t$, and the probability that any particular transcript occurs in any given session is $1/(q\,2^t)$.
- Oscar, say, can generate simulated transcripts, having the same probability distribution, **without taking part in the protocol**:
  1. choose $r \in_R \{1, \ldots, 2^t\}$
  2. choose $y \in_R \{0, \ldots, q-1\}$
  3. compute $\gamma = \alpha^y v^r \bmod p$
- $\mathsf{Pr}_{\mathsf{sim}}[T] = \mathsf{Pr}_{\mathsf{real}}[T] = 1/(q\,2^t)$ for all $T \in \mathcal{T}$

Security wrt an Honest Verifier

- Whatever an honest verifier can compute after taking part in several sessions of the protocol, Oscar can compute without taking part in any sessions of the protocol.
- In particular, computing Alice's private key, a, which is necessary for Oscar to be able to impersonate Alice, is not made easier for Oscar if he plays the role of the verifier in one or more sessions and he (Oscar) chooses his challenges randomly.
- Is it possible that Oscar can obtain some useful information by choosing his challenges r in a **non-uniform way** (depending on γ, say)?
- This does not seem likely, but the only known security proofs of the *Schnorr scheme* for arbitrary verifiers require additional assumptions.

Summary of Properties of the Schnorr Scheme

- The protocol is a **proof of knowledge**: breaking the protocol is equivalent to computing the private key.
- Transcripts of sessions can be perfectly simulated (i.e., generated at random, without taking part in the protocol), assuming that the challenges r are chosen uniformly at random.
- This implies that no information about the private key is revealed to an honest verifier, and the protocol is said to be **honest verifier zero-knowledge**.
- Therefore, assuming that computing the discrete logarithm a is infeasible, the protocol is secure with respect to an honest verifier.
- Note: a dishonest verifier may choose challenges r depending on γ, and the resulting probability distribution on the set of possible transcripts apparently cannot be simulated.

Simulatable Probability Distributions

- Why does simulatability imply zero-knowledge?
- Suppose there exists an algorithm *Extract* which, when given a set of transcripts, say $T_1, \ldots, T_\ell$, computes a private key, say a, with some probability, say ϵ.
- We assume that the transcripts are actual transcripts of sessions, in which the participants follow the protocol.
- Suppose that $T'_1, \ldots, T'_\ell$ are simulated transcripts.
- The probability distribution on simulated transcripts is **identical** to the probability distribution on real transcripts.
- Therefore *Extract*$(T'_1, \ldots, T'_\ell)$ will also compute a with the same probability ϵ.
- Executing the protocol does not make computing a easier, so the protocol is zero-knowledge.

What is Two-channel Cryptography?

- Two channels are accessible for communication. They have different properties in terms of security and cost.
- **broadband insecure channel**: wireless channel,
- **narrow-band authenticated channel**: voice, data comparison, data imprinting, near field communication, visible light, infra red signals, laser. This is sometimes termed a **manual channel**, meaning, for example, that Alice types a one-time password into a terminal.
- **Goal:** to achieve a certain cryptographic goal by using the two channels while optimizing the computational and communication complexity.

Two-channel Cryptography in Ad Hoc Networks

- An **ad hoc network** is spontaneous: The connection is established for the duration of one session.
- It should be easy to quickly add new users and remove users.
- Secret-key techniques may not be not practical (no prior shared secret keys exist).
- Public-key techniques may be too expensive (computationally) for constrained platforms.
- Identity-based systems also need some infrastructure.
- What can we do in absence of a public or secret key?
- **Two-channel Authentication!**

Authentication using Two-Channel Cryptography

- The focus is on **message authentication in ad hoc networks**.
- We assume a totally insecure **broadband channel**: $\rightarrow$. The broadband channel can be used to send long messages.
- As well, we have a "moderately secure" **narrow-band channel**: $\Rightarrow$. The narrow-band channel can be used to authenticate messages.
- Two small devices, Alice and Bob, wish to establish a secure key, M, in the presence of an active adversary, Oscar.
- Oscar has **full** control over the broadband channel.
- Oscar **cannot modify a message or initiate a new flow** over the narrow-band channel.

Message Authentication Protocols (MAPs)

- Alice wants to authenticate a message, $M \in \mathcal{M}$, to Bob (along with her identity).
- It should be the case that once the *MAP* is carried out, either Bob rejects or he outputs (Alice, M'), where $M' \in \mathcal{M}$.
- If there is no active adversary, then $M = M'$.

Adversarial Goal:

- Oscar is trying to make Bob accept a message M' along with the identity of Alice, when Alice has never sent M'.
- In a successful attack, Bob outputs (Alice, M'), where Alice has never sent M'.

Non-interactive MAP (NIMAP)

A typical flow structure:

Alice		Bob
Input (M, Bob)		
	$\longrightarrow$	
	$\Longrightarrow$	
		Output (Alice, M') or reject.

Interactive MAP (IMAP)

A possible flow structure:

Alice		Bob
Input (M, Bob)	$\longrightarrow$	
	$\longleftarrow$	
	$\longrightarrow$	
	$\Longrightarrow$	Output (Alice, M') or reject.

Mashatan-Stinson NIMAP

Let H be a **hash function** (which satisfies a certain property, to be defined later).

Alice		Bob
Input (M, Bob), $\|M\| = \ell_1$,		
Choose $K \in_R \{0,1\}^{\ell_2}$	$\xrightarrow{M, K}$	Receive M', K'
Compute $h = H(M \| K)$	$\xrightarrow{h}$	Receive h
		Accept if $h = H(M' \| K')$, reject otherwise

Note: Oscar can modify M and/or K but he cannot modify h.

The Attack Scenario

Alice		Oscar		Bob
	$\xleftarrow{M}$	$M \in \{0,1\}^{\ell_1}$		
$K \in_R \{0,1\}^{\ell_2}$	$\xrightarrow{M,K}$	$\lvert M' \rvert = \ell_1, \lvert K' \rvert = \ell_2$	$\xrightarrow{M',K'}$	
$h = H(M \Vert K)$	$\xRightarrow{h}$		$\xRightarrow{h}$	$h = H(M' \Vert K')?$

Hybrid-Collision Resistance

Definition

A **hybrid-collision resistant (HCR) hash function**, H, is a hash function where the following **HCR Game** is hard to win. The pair $(L, M\|K)$ is a **hybrid-collision**.

Oscar		Challenger
Choose M, $\|M\| = \ell_1$.	$\xrightarrow{M}$	Choose $K \in_R \{0,1\}^{\ell_2}$.
	$\xleftarrow{K}$	
Choose L, $\|L\| = \ell_1 + \ell_2$.	$\xrightarrow{L}$	Oscar wins if $L \neq M\|K$ and $H(M\|K) = H(L)$.

If an adversary with computational complexity T wins the **HCR game** with probability **at most** ϵ, then H is a (T, ϵ)-**HCR** hash function.

Hardness of the HCR Game

- We analyze the **HCR Game** in the **random oracle model**.
- Let H be a random oracle, i.e., a hash function having domain $\mathcal{X}$ and range $\mathcal{Y}$, where $|\mathcal{Y}| = 2^d$, that produces **completely random outputs** in $\mathcal{Y}$.
- Assume that Oscar is only permitted oracle access to H (in particular, there is no algorithm describing H).
- Oscar can query the oracle H at most $T = 2^t$ times, where $t < \min\{d, \ell_2\}$.
- Let ϵ be the probability of Oscar winning the **HCR Game**.
- We will show that

$$\epsilon \leq 2^{t-d} + 2^{2t-d-\ell_2}.$$

Analysis of the HCR Game

- Suppose Oscar makes S oracle queries, say $H(X_1), \ldots, H(X_S)$, before sending M to the Challenger.
- Then Oscar makes $T - S$ additional oracle queries, say $H(X_{S+1}), \ldots, H(X_T)$, after receiving K from the Challenger.
- We can assume that $L \in \{X_1, \ldots, X_T\}$ and $M \| K \in \{X_1, \ldots, X_T\}$ (otherwise, $\epsilon \leq 2^{-d}$).
- We divide the analysis into three cases:
  1. $L \in \{X_{S+1}, \ldots, X_T\}$.
  2. $L \in \{X_1, \ldots, X_S\}$ and $M \| K \in \{X_{S+1}, \ldots, X_T\}$
  3. $L \in \{X_1, \ldots, X_S\}$ and $M \| K \in \{X_1, \ldots, X_S\}$

Analysis of the HCR Game (cont.)

- In **case 1**, $M \| K$ is determined before the query $H(L)$ is made.
- For any $i \in \{S+1, \ldots, T\}$ the probability that $H(X_i) = H(M \| K)$ is 2^{-d}.
- Therefore, the probability of a collision occurring in **case 1** is at most

$$\frac{T-S}{2^d}.$$

- In **case 2**, $H(M \| K)$ collides with one of the previously computed values $H(X_i)$, where $i \in \{1, \ldots, S\}$.
- Therefore, the probability of a collision occurring in **case 2** is at most

$$\frac{S}{2^d}.$$

Analysis of the HCR Game (cont.)

- In **case 3**, the probability that $M \| K \in \{X_1, \ldots, X_S\}$ is at most $S/2^{\ell_2}$
- Given that $M \| K \in \{X_1, \ldots, X_S\}$, say $M \| K = X_i$ where $1 \leq i \leq S$, the probability that X_i collides with some X_j ($j \neq i$, $1 \leq j \leq S$) is at most $(S-1)/2^d$.
- Therefore, the probability of a collision occurring in **case 3**, is at most

$$\frac{S}{2^{\ell_2}} \times \frac{S-1}{2^d} \leq \frac{T}{2^{\ell_2}} \times \frac{T}{2^d} = 2^{2t-d-\ell_2}.$$

- The probability of a collision is at most the **sum** of the probabilities of **case 1**, **case 2** and **case 3**, namely,

$$\frac{T-S}{2^d} + \frac{S}{2^d} + 2^{2t-d-\ell_2} = 2^{t-d} + 2^{2t-d-\ell_2}.$$

Security of the NIMAP

Theorem

Let H be a (T, ϵ)-HCR hash function. Any adversary against the Mashatan-Stinson NIMAP having complexity $T = 2^t$, has a probability of success that is at most ϵ.

Typical parameter choices:

- $d = 80$ (size of the message digest, i.e., the # of bits sent over the authenticated channel),
- $t = 50$ (i.e., the adversary makes 2^{50} oracle queries).
- $\ell_2 = 60$ (the number of random bits supplied by Alice in the protocol)
- If we treat the hash function as a random oracle, we have

$$\epsilon \leq 2^{t-d} + 2^{2t-d-\ell_2} = 2^{-30} + 2^{-40} \approx 2^{-30}.$$

Table of Contents

Key Distribution, Transport and Agreement Protocols

- There are many possible scenarios, including the following:
  - a TA distributes keying information "ahead of time" to everyone in the network (**key predistribution scheme** or **KPS**)
  - an online TA distributes session keys upon request of network users via an interactive protocol (**session key distribution scheme** or **SKDS**, e.g., *Kerberos*)
  - network users employ interactive protocols to construct (session) keys (**key agreement scheme** or **KAS**)
- as always, we can consider schemes and protocols in the public/private key setting or the secret key setting; active and/or passive adversaries; various adversarial goals, attack models, security levels, etc.

Long-lived Keys and Session Keys

- users (or pairs of users) may have **long-lived keys** (**LL-keys**) which should be pre-computed and stored securely; or computed (non-interactively) as needed from each user's secret information
- pairs of users will employ short-lived **session keys** in a particular session and then throw them away
- LL-keys are generally used in protocols to generate session keys
- a KPS provides a method to distribute secret LL-keys; then a KAS is used to generate session keys as needed
- a SKDS is a three-party protocol, usually based on long-lived secret keys held by each user and the TA
- a KAS can be secret-key based or public/private key based

Why use Session Keys?

The following list is paraphrased from the **Handbook of Applied Cryptography**:

- to limit the amount of ciphertext (encrypted with one particular key) available to an attacker
- to limit exposure in the event of session key compromise (i.e. the compromise of a session key should not compromise the LL-key, or other session keys)
- to (possibly) reduce the amount of long-term information that needs to be securely stored by each user, by generating keys only when they are needed
- to create independence across sessions or applications

Requirements of Long-lived Keys

- assume a network of n users
- the "type" of scheme used to construct session keys dictates the type of LL-keys required
- users' storage requirements depend on the type of keys used
- SKDS requires that each network user have a secret LL-key in common with the TA:
  - **low storage requirement** for network users
  - an **on-line** TA is required for session key distribution

Requirements of Long-lived Keys (cont.)

- secret-key based KAS requires that every pair of network users has a secret LL-key known only to them
  - **high storage requirement**: each user stores $n-1$ keys (and there are $\binom{n}{2}$ keys in total)
  - a TA is **not required** for session key distribution
- public-key based KAS requires that each network user have his/her own public/private LL-key pair
  - **low storage requirement**: each user stores one private key and a certificate
  - a TA is **not required** for session key distribution
  - an **offline** TA (or CA) is required to maintain public-key infrastructure

Overview of Attack Models and Adversarial Goals for SKDS and KAS

- we consider passive and active adversaries (as in identification protocols)
- we usually require **authenticated** SKDS and KAS, which include (mutual) identification of Alice and Bob
  - the protocols should be secure identification protocols
  - in addition, Alice and Bob should possess a new secret key at the end of the protocol, whose value is not known to the adversary

Extended Attack Models and Adversarial Goals

Extended attack models are also considered, such as:

- Suppose that the adversary learns the value of a particular session key (this is called a **known session key attack**).
  - In this attack model, we would still want other session keys (as well as the LL-keys) to remain secure.
- Suppose that the adversary learns the LL-keys of the participants (this is a **known LL-key attack**).
  - This is a catastrophic attack: a new scheme must be set up.
  - Can we limit the damage done? If the adversary cannot learn the values of previous session keys, then the scheme possesses the property of **perfect forward secrecy**.

Attack Models and Adversarial Goals for KPS

- we assume that the TA distributes secret information securely to the network users
- the adversary may corrupt a subset of at most k users, and obtain all their secret information, where k is a **security parameter**
- the adversary's goal is to determine the secret LL-key of a pair of uncorrupted users
- the *Blom KPS* is a KPS that is unconditionally secure against adversaries of this type
- suppose that $p \geq n$ is prime
- LL-keys will be elements of $\mathbb{Z}_p$
- each user must store $k+1$ elements of $\mathbb{Z}_p$ as his/her secret information (the storage requirement is independent of n)

The Blom PKS ($k = 1$)

For each user U, a value r_U is made public. The values r_U must be distinct elements of $\mathbb{Z}_p$.

Protocol: *Blom's key distribution scheme ($k = 1$)*

1. The TA chooses three random elements $a, b, c \in \mathbb{Z}_p$ (not necessarily distinct), and forms the polynomial

$$f(x, y) = a + b(x + y) + cxy \bmod p.$$

2. For each user U, the TA computes the polynomial

$$g_U(x) = f(x, r_U) \bmod p = a_U + b_U x$$

and transmits (a_U, b_U) to U over a secure channel.

The Blom PKS (cont.)

- the LL-key for U and V is

$$K_{U,V} = K_{V,U} = f(r_U, r_V),$$

where U computes $K_{U,V} = g_U(r_V)$ and V computes $K_{U,V} = g_V(r_U)$

- we have:

$$\begin{aligned} a_U &= a + b\,r_U \bmod p \quad \text{and} \\ b_U &= b + c\,r_U \bmod p, \quad \text{so} \\ g_U(r_V) &= a + b\,r_U + (b + c\,r_U)r_V \\ &= a + b(r_U + r_V) + c\,r_U r_V \\ &= f(r_U, r_V) \bmod p \end{aligned}$$

A Toy Example

- suppose $p = 17$
- suppose there are three users: U, V and W, and their public values are $r_U = 12$, $r_V = 7$ and $r_W = 1$
- suppose the TA chooses $a = 8$, $b = 7$ and $c = 2$, so the polynomial f is

$$f(x, y) = 8 + 7(x + y) + 2xy$$

- the g polynomials are as follows:

$$\begin{aligned} g_U(x) &= 7 + 14x \\ g_V(x) &= 6 + 4x \\ g_W(x) &= 15 + 9x \end{aligned}$$

A Toy Example

- the three keys are

$$\begin{aligned} K_{U,V} &= 3 \\ K_{U,W} &= 4 \\ K_{V,W} &= 10 \end{aligned}$$

- U would compute $K_{U,V}$ as

$$g_U(r_V) = 7 + 14 \times 7 \bmod 17 = 3$$

- V would compute $K_{U,V}$ as

$$g_V(r_U) = 6 + 4 \times 12 \bmod 17 = 3$$

Unconditional Security of the Blom Scheme ($k = 1$)

- we show that no one user, say W, can determine any information about a key for two other users, say $K_{U,V}$
- what information does W possess?
- W knows the values

$$a_W = a + b\, r_W \bmod p$$

and

$$b_W = b + c\, r_W \bmod p$$

- the key that W is trying to compute is

$$K_{U,V} = a + b(r_U + r_V) + c\, r_U r_V \bmod p.$$

- r_U, r_V and r_W are public, but a, b and c are unknown

Security of the Blom Scheme (cont.)

- we will show that the information known by W is consistent with any possible value $K^* \in \mathbb{Z}_p$ of the key $K_{U,V}$
- consider the following matrix equation (in $\mathbb{Z}_p$):

$$\begin{pmatrix} 1 & r_U + r_V & r_U r_V \\ 1 & r_W & 0 \\ 0 & 1 & r_W \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} K^* \\ a_W \\ b_W \end{pmatrix}.$$

- the determinant of the coefficient matrix is

$$r_W{}^2 + r_U r_V - (r_U + r_V) r_W = (r_W - r_U)(r_W - r_V),$$

  where all arithmetic is done in $\mathbb{Z}_p$
- since $r_W \neq r_U$ and $r_W \neq r_V$, it follows that the coefficient matrix has non-zero determinant, and hence the matrix equation has a unique solution for a, b and c

Security of the Blom Scheme (cont.)

- a coalition of two users, say $\{W, X\}$, will be able to compute **any** key $K_{U,V}$ where $\{W, X\} \cap \{U, V\} = \emptyset$
- W and X together have the following information:

$$\begin{aligned} a_W &= a + b\,r_W \\ b_W &= b + c\,r_W \\ a_X &= a + b\,r_X \\ b_X &= b + c\,r_X \end{aligned}$$

  where a, b and c are unknowns
- W and X together have four equations in three unknowns, and they can easily compute a unique solution for a, b and c
- once they know a, b and c, they can form the polynomial $f(x, y)$ and compute any key they wish

Protocol: *Blom's key distribution scheme (arbitrary k)*

1. For $0 \leq i, j \leq k$, the TA chooses random elements $a_{i,j} \in \mathbb{Z}_p$, such that $a_{i,j} = a_{j,i}$ for all i, j. Then the TA forms the polynomial

$$f(x, y) = \sum_{i=0}^{k} \sum_{j=0}^{k} a_{i,j}\, x^i y^j \bmod p.$$

2. For each user U, the TA computes the polynomial

$$g_U(x) = f(x, r_U) \bmod p = \sum_{i=0}^{k} a_{U,i}\, x^i$$

   and transmits the coefficient vector $(a_{U,0}, \ldots, a_{U,k})$ to U over a secure channel.

3. For any two users U and V, the key $K_{U,V} = f(r_U, r_V)$.

Security of the Blom Scheme (arbitrary k)

The Blom scheme satisfies the following security properties:

1. no set of k users, say $W_1, \ldots, W_k$ can determine any information about a key for two other users, say $K_{U,V}$
2. any set of $k+1$ users, say $W_1, \ldots, W_{k+1}$, can break the scheme

Security of the Blom Scheme (cont.)

- a set of users $W_1, \ldots, W_\ell$ (collectively) know the polynomials

$$g_{W_i}(x) = f(x, r_{W_i}) \bmod p,$$

  $1 \leq i \leq \ell$
- we use a **Lagrange interpolation formula** to prove 2.
- suppose p is prime; $x_1, x_2, \ldots, x_{m+1} \in \mathbb{Z}_p$ are distinct; and $a_1, a_2, \ldots, a_{m+1} \in \mathbb{Z}_p$ are not necessarily distinct
- there is a unique polynomial $A(x) \in \mathbb{Z}_p[x]$ having degree at most m such that $A(x_i) = a_i$, $1 \leq i \leq m+1$
- the polynomial $A(x)$ is defined as follows:

$$A(x) = \sum_{j=1}^{m+1} a_j \prod_{1 \leq h \leq m+1, h \neq j} \frac{x - x_h}{x_j - x_h}.$$

Bivariate Polynomial Interpolation

- the Lagrange interpolation formula has a **bivariate form**:
- suppose p is prime; $y_1, y_2, \ldots, y_{m+1} \in \mathbb{Z}_p$ are distinct; and suppose that $a_1(x), a_2(x), \ldots, a_{m+1}(x) \in \mathbb{Z}_p[x]$ are polynomials of degree at most m
- there is a unique polynomial $A(x, y) \in \mathbb{Z}_p[x, y]$ having degree at most m (in x and y) such that $A(x, y_i) = a_i(x)$, $1 \leq i \leq m + 1$
- the polynomial $A(x, y)$ is defined as follows:

$$A(x, y) = \sum_{j=1}^{m+1} a_j(x) \prod_{1 \leq h \leq m+1, h \neq j} \frac{y - y_h}{y_j - y_h}.$$

Example of Bivariate Interpolation

Suppose that $p = 13$, $m = 2$, $y_1 = 1$, $y_2 = 2$, $y_3 = 3$ $a_1(x) = 1 + x + x^2$, $a_2(x) = 7 + 4x^2$ and $a_3(x) = 2 + 9x$.

$$\begin{aligned}
\frac{(y-2)(y-3)}{(1-2)(1-3)} &= 7y^2 + 4y + 3 \\
\frac{(y-1)(y-3)}{(2-1)(2-3)} &= 12y^2 + 4y + 10 \\
\frac{(y-1)(y-2)}{(3-1)(3-2)} &= 7y^2 + 5y + 1
\end{aligned}$$

$$\begin{aligned}
A(x,y) &= (1 + x + x^2)(7y^2 + 4y + 3) + (7 + 4x^2)(12y^2 + 4y + 10) \\
&\quad +(2 + 9x)(7y^2 + 5y + 1) \bmod 13 \\
&= y^2 + 3y + 10 + 5xy^2 + 10xy + 12x + 3x^2y^2 + 7x^2y + 4x^2
\end{aligned}$$

Insecurity wrt $k+1$ Colluders

- a set of bad users $W_1, \ldots, W_{k+1}$ (collectively) know the polynomials

$$g_{W_i}(x) = f(x, r_{W_i}) \bmod p,$$

  $1 \leq i \leq k+1$
- using the bivariate interpolation formula, they can compute $f(x, y)$
- then they can compute any key

Security wrt k Colluders

- a set of bad users $W_1, \ldots, W_k$ (collectively) know the polynomials

$$g_{W_i}(x) = f(x, r_{W_i}) \bmod p,$$

  $1 \leq i \leq k$
- we show that this information is consistent with any possible value of the key
- let K be the real (unknown) key, and let $K^* \neq K$
- define a polynomial $f^*(x, y)$ as follows:

$$f^*(x, y) = f(x, y) + (K^* - K) \prod_{1 \leq i \leq k} \frac{(x - r_{W_i})(y - r_{W_i})}{(r_U - r_{W_i})(r_V - r_{W_i})}$$

Security wrt k Colluders (cont.)

- f^* is a symmetric polynomial (i.e., $f(x,y) = f(y,x)$)
- for $1 \leq i \leq k$, it holds that

$$f^*(x, r_{W_i}) = f(x, r_{W_i}) = g_{W_i}(x)$$

- further,

$$f^*(r_U, r_V) = f(r_U, r_V) + K^* - K = K^*$$

- For any possible value of the key, K^*, there is a symmetric polynomial f^* such that the key $K_{U,V} = K^*$ and such that the secret information held by the k bad users is unchanged

Subgroups of Cyclic Groups (review)

- suppose that $(G, \cdot)$ is a cyclic group of order n
- let α be a generator of G (i.e., $\text{ord}(\alpha) = n$)
- suppose that m is a divisor of n
- there is a unique subgroup H of G having order m
- the subgroup H is cyclic, and $\alpha^{n/m}$ is a generator of H (i.e., $\text{ord}(\alpha^{n/m}) = m$)
- H consists of all the elements of G that have order dividing m
- if m is prime, then all elements of H other than the identity have order m (and hence they are all generators of H)

The Diffie-Hellman KPS

- the *Diffie-Hellman KPS* is a public-key based scheme to distribute secret LL-keys
- suppose α is an element having prime order q in the group ${\mathbb{Z}_p}^*$, where p is prime, $p - 1 \equiv 0 \pmod{q}$, $p \approx 2^{1024}$ and $q > 2^{160}$
- α, p and q are public domain parameters
- every user U has a private LL-key a_U (where $0 \leq a_U \leq q - 1$) and a corresponding public key

$$b_U = \alpha^{a_U} \bmod p$$

- the users' public keys are signed by the TA and stored on certificates, as usual

The Diffie-Hellman KPS (cont.)

- the secret LL-key $K_{U,V}$ for two users U and V is defined as follows:

$$K_{U,V} = \alpha^{a_U a_V} \bmod p$$

- V computes

$$K_{U,V} = {b_U}^{a_V} \bmod p,$$

  using the public key b_U from U's certificate, together with his own secret key a_V
- U computes

$$K_{U,V} = {b_V}^{a_U} \bmod p,$$

  using the public key b_V from V's certificate, together with her own secret key a_U

Security of the Diffie-Hellman KPS

- a coalition of bad users is of no help to the adversary in determining the key belonging to some disjoint pair of users
- the adversary's attempt to compute a key $K_{U,V}$ is an instance of the **Computational Diffie-Hellman** problem:

Problem

Computational Diffie-Hellman *(**CDH**)*

Instance: *A multiplicative group $(G, \cdot)$, an element $\alpha \in G$ having order n, and two elements $\beta, \gamma \in \langle \alpha \rangle$.*

Question: *Find $\delta \in \langle \alpha \rangle$ such that*

$$\log_\alpha \delta \equiv \log_\alpha \beta \times \log_\alpha \gamma \pmod{n}.$$

(Equivalently, given $\beta = \alpha^b$ and $\gamma = \alpha^c$, where b and c are unknown, compute $\delta = \alpha^{bc}$.)

Computational Diffie-Hellman $\propto_T$ Discrete Logarithm

- the **Computational Diffie-Hellman** problem is no harder to solve than the **Discrete Logarithm** problem in the same subgroup $\langle \alpha \rangle$
- given an oracle for the **DLP**, it is easy to solve the **CDH** problem, as follows:
- given inputs α, β, γ for **CDH**,
  1. use the oracle to compute $b = \log_\alpha \beta$
  2. compute $\delta = \gamma^b$
- the **Computational Diffie-Hellman** problem is thought to be infeasible when $G = \mathbb{Z}_p$ where $p \approx 2^{1024}$ is prime, n is a divisor of $p - 1$, and n has *at least one prime divisor* q *with* $q > 2^{160}$

Partial Information about Diffie-Hellman Keys

- the adversary may be unable to compute a Diffie-Hellman key but he could still (possibly) determine some partial information about the key
- we desire **semantic security** of the keys, which means that an adversary can compute no partial information about them (in polynomial time, say)
- in other words, distinguishing Diffie-Hellman keys from random elements of the subgroup $\langle \alpha \rangle$ should be infeasible
- semantic security of Diffie-Hellman keys is equivalent to the infeasability of the **Decision Diffie-Hellman** problem

The Decision Diffie-Hellman Problem

Problem

Decision Diffie-Hellman *(**DDH**)*

Instance: *A multiplicative group* $(G, \cdot)$*, an element* $\alpha \in G$ *having order* n*, and three elements* $\beta, \gamma, \delta \in \langle \alpha \rangle$*.*

Question: *Is it the case that* $\log_\alpha \delta \equiv \log_\alpha \beta \times \log_\alpha \gamma \pmod{n}$*? (Equivalently, given* α^b*,* α^c *and* α^d*, where* b, d *and* d *are unknown, determine if* $d \equiv bc \pmod{n}$*.)*

- It is easy to see that the **Decision Diffie-Hellman** problem is no harder to solve than the **Computational Diffie-Hellman** problem in the same subgroup $\langle \alpha \rangle$

Decision Diffie-Hellman $\propto_T$ Computational Diffie-Hellman

- given an oracle for **CDH**, it is easy to solve the **DDH** problem, as follows:
- given inputs $\alpha, \beta, \gamma, \delta$ for **DDH**,
  1. use the oracle to to find the value δ' such that

$$\log_\alpha \delta' \equiv \log_\alpha \beta \times \log_\alpha \gamma \pmod{n}$$

  2. check to see if $\delta' = \delta$

Security of DDH

- the **Decision Diffie-Hellman** problem is thought to be infeasible when $G = \mathbb{Z}_p$ where $p \approx 2^{1024}$ is prime, n is a divisor of $p - 1$, and n has **no prime divisor q with $q < 2^{160}$**
- this is a stronger condition than the one that is conjectured for the security of the **Computational Diffie-Hellman** problem

Wireless Sensor Networks

- **sensor nodes** have limited computation and communication capabilities
- a **wireless sensor network** (or **WSN**) of 1000 – 10000 sensor nodes is distributed in a random way in a possibly hostile physical environment
- the sensor nodes operate unattended for extended periods of time
- the sensor nodes have no external power supply, so they should consume as little battery power as possible
- usually, the sensor nodes communicate using secret key cryptography
- a set of secret keys is installed in each node, before the sensor nodes are deployed, using a suitable key predistribution scheme (or KPS)
- nodes may be stolen by an adversary (this is called **node compromise**)

Two Trivial Schemes

1. If every node is given the same secret **master key**, then memory costs are low. However, this situation is unsuitable because the compromise of a single node would render the network completely insecure.
2. For every pair of nodes, there could be a secret **pairwise key** given only to these two nodes. This scheme would have optimal resilience to node compromise, but memory costs would be prohibitively expensive for large networks because every node would have to store $n - 1$ keys, where n is the number of nodes in the WSN.

Eschenauer-Gligor and Related Schemes

- In 2002, Eschenauer and Gligor proposed a **probabilistic** approach to key predistribution for sensor networks. For a suitable value of k, every node is assigned a random k-subset of keys chosen from a given pool of secret keys.
- In 2003, Chan, Perrig and Song suggested that two nodes should compute a pairwise key only if they share at least η common keys, where the integer $\eta \geq 1$ is a pre-specified **intersection threshold**. Such a pair of nodes is termed a **link**.
- Suppose that U_i and U_j have exactly $\ell \geq \eta$ common keys, say $key_{a_1}, \ldots, key_{a_\ell}$, where $a_1 < a_2 < \cdots < a_\ell$. Then they can each compute the same pairwise secret key,

$$K_{i,j} = h(key_{a_1} \parallel \ldots \parallel key_{a_\ell} \parallel i \parallel j),$$

using a **key derivation function** h that is constructed from a secure public hash function, e.g., *SHA-1*.

Attack Model

- The most studied adversarial model in WSNs is **random node compromise**.
- An adversary compromises a fixed number of **randomly chosen nodes** in the network and extracts the keys stored in them.
- Any links involving the compromised nodes are broken.
- However, this can also cause other links to be broken that do not directly involve the compromised nodes.
- A link formed by two nodes A_1, A_2, where $|A_1 \cap A_2| \geq \eta$, will be **broken** if a node $B \notin \{A_1, A_2\}$ is compromised, provided that $A_1 \cap A_2 \subseteq B$.
- If s nodes, say $B_1, \ldots, B_s$, are compromised, then a link A_1, A_2 will be broken whenever

$$A_1 \cap A_2 \subseteq \bigcup_{i=1}^{s} B_i.$$

Important Metrics

Storage requirements

The number of keys stored in each node, which is denoted by k, should be "small" (e.g., at most 100).

Network connectivity

The probability that a randomly chosen pair of nodes can compute a common key is denoted by Pr_1. Pr_1 should be "large" (e.g., at least 0.5).

Network resilience

The probability that a random link is broken by the compromise of s randomly chosen nodes not in the link is denoted by $fail(s)$. We want $fail(s)$ to be small: **high resilience** corresponds to a **small value** for $fail(s)$. For simplicity, we just consider $fail(1)$ here.

Remark: As η is increased, Pr_1 and $fail(1)$ both decrease.

Deterministic Schemes

- In 2004, **deterministic KPS** were proposed independently by Camtepe and Yener; by Lee and Stinson; and by Wei and Wu.
- A suitable **set system** is chosen, and each block is assigned to a node in the WSN (the design and the correspondence of nodes to blocks is **public**).
- The points in the block are the **indices** of the keys given to the corresponding node.
- Probabilistic schemes are analyzed using **random graph theory**, and desirable properties hold with **high probability**.
- Deterministic schemes can be **proven** to have desirable properties, and they have more efficient algorithms for **shared-key discovery** than probabilistic schemes.

Combinatorial Set Systems (aka Designs)

- A **set system** is a pair $(X, \mathcal{A})$, where the elements of X are called **points** and $\mathcal{A}$ is a set of subsets of X, called **blocks**.
- We pair up the blocks of the set system with the nodes in the WSN.
- The points in the block are the **key identifiers** of the keys given to the corresponding node.
- The **degree** of a point $x \in X$ is the number of blocks containing x
- $(X, \mathcal{A})$ is **regular** (of degree r) if all points have the same degree, r; then each key occurs in r nodes in the WSN.
- If all blocks have size k, then $(X, \mathcal{A})$ is said to be **uniform** (of rank k); then each node is assigned k keys.
- A (v, b, r, k)-**configuration** is a set system $(X, \mathcal{A})$ where $|X| = v$ and $|\mathcal{A}| = b$, that is uniform of rank k and regular of degree r, such that every pair of points occurs in at most one block.
- In a configuration, it holds that $vr = bk$.

Toy Example

We list the blocks in a $(7, 7, 3, 3)$-configuration (a projective plane of order 2) and the keys in a corresponding KPS:

node	block	key assignment
N_1	$\{1, 2, 4\}$	k_1, k_2, k_4
N_2	$\{2, 3, 5\}$	k_2, k_3, k_5
N_3	$\{3, 4, 6\}$	k_3, k_4, k_6
N_4	$\{4, 5, 7\}$	k_4, k_5, k_7
N_5	$\{1, 5, 6\}$	k_1, k_5, k_6
N_6	$\{2, 6, 7\}$	k_2, k_6, k_7
N_7	$\{1, 3, 7\}$	k_1, k_3, k_7

The actual values of keys are **secret**, but the lists of key identifiers (i.e., the blocks) are **public**.
In this example, $Pr_1 = 1$ and $fail(1) = 1/5$.

Properties of Configuration-based KPS

- For a configuration-based KPS, we take $\eta = 1$.
- Every block intersects $k(r-1)$ blocks in one point and is disjoint from all the other blocks.
- Therefore
$$Pr_1 = \frac{k(r-1)}{b-1}.$$
- A link L is defined by **two blocks** that intersect in one point, say x.
- There are $r-2$ other blocks that contain x; the corresponding nodes will compromise the link L.
- Therefore,
$$fail(1) = \frac{r-2}{b-2}.$$
- There is a tradeoff between Pr_1 and $fail(1)$, which is quantified by computing the ratio $\rho = Pr_1/fail(1)$:
$$\rho = \frac{k(b-2)(r-1)}{(b-1)(r-2)} \approx k.$$

Transversal Designs

- Lee and Stinson (2005) proposed using transversal designs to construct KPS.
- Let n, k and t be positive integers
- A **transversal design TD**(t, k, n) is a triple $(X, \mathcal{H}, \mathcal{A})$, where X is a finite set of cardinality kn, $\mathcal{H}$ is a partition of X into k parts (called **groups**) of size n, and $\mathcal{A}$ is a set of k-subsets of X (called **blocks**), which satisfy the following properties:
  1. $|H \cap A| = 1$ for every $H \in \mathcal{H}$ and every $A \in \mathcal{A}$, and
  2. every t elements of X from different groups occurs in exactly one block in $\mathcal{A}$.
- **Bose-Bush bound**: When $t = 2, 3$, a TD(t, k, n) exists only if $k \leq n + t - 1$.

An Easy Construction for Transversal Designs

- Suppose that p is prime and $t \leq k \leq p$.
- Define
$$X = \{0, \ldots, k-1\} \times \mathbb{Z}_p.$$
- For every **ordered t-subset** $\mathbf{c} = (c_0, \ldots, c_{t-1}) \in (\mathbb{Z}_p)^t$, define a block
$$A_{\mathbf{c}} = \left\{ \left(x, \sum_{i=0}^{t-1} c_i x^i \right) : 0 \leq x \leq k-1 \right\}.$$
- Let
$$\mathcal{A} = \{A_{\mathbf{c}} : \mathbf{c} \in (\mathbb{Z}_p)^t\}.$$
- Then $(X, \mathcal{A})$ is a TD(t, k, p).
- The construction can be adapted to any finite field $\mathbb{F}_q$, where q is a prime power.
- These transversal designs are equivalent to **Reed-Solomon codes**.

Example

Suppose we take $p = 5$ and $k = 4$; then we construct a TD$(2, 4, 5)$:

$A_{0,0}$={00,10,20,30} $A_{0,1}$={01,11,21,31} $A_{0,2}$={02,12,22,32}
$A_{0,3}$={03,13,23,33} $A_{0,4}$={04,14,24,34} $A_{1,0}$={00,11,22,33}
$A_{1,1}$={01,12,23,34} $A_{1,2}$={02,13,24,30} $A_{1,3}$={03,14,20,31}
$A_{1,4}$={04,14,24,34} $A_{2,0}$={00,12,24,31} $A_{2,1}$={01,13,20,32}
$A_{2,2}$={02,14,21,33} $A_{2,3}$={03,10,22,34} $A_{2,4}$={04,11,23,30}
$A_{3,0}$={00,13,21,34} $A_{3,1}$={01,14,22,30} $A_{3,2}$={02,10,23,31}
$A_{3,3}$={03,11,24,32} $A_{3,4}$={04,12,20,33} $A_{4,0}$={00,14,23,32}
$A_{4,1}$={01,10,24,33} $A_{4,2}$={02,11,20,34} $A_{4,3}$={03,12,21,30}
$A_{4,4}$={04,13,22,31}

Properties of KPS from TDs with $t = 2$

- A TD$(2, k, n)$ is an (nk, n^2, n, k)-configuration.
- Therefore

$$Pr_1 = \frac{k(n-1)}{n^2-1} = \frac{k}{n+1} \quad \text{and} \quad fail(1) = \frac{n-2}{n^2-2}.$$

- Since the set system is a configuration, we have $\rho \approx k$.

Benefit:

We can make Pr_1 arbitrarily close to 1.

Drawback:

The **network size** is n^2, which may not be large enough for "reasonable" values of n.

Drawback:

The ratio $\rho \approx k$ is a bit small for many applications (this applies to any configuration-based KPS).

Properties of KPS from TDs with $t = 3$, $\eta = 2$

- We can base a KPS on a TD$(3, k, n)$ with $\eta = 1$ or 2.
- When $\eta = 2$, we have

$$Pr_1 = \frac{k(k-1)}{2(n^2+n+1)} \quad \text{and} \quad fail(1) = \frac{n-2}{n^3-2}.$$

Drawback:

The maximum value of Pr_1 is about $1/2$.

Benefit:

The **network size** is n^3, which is quite large, even for "reasonable" values of n.

Benefit:

The ratio $\rho \approx k^2/2$ is now considerably larger.

Session Key Distribution

- The TA shares secret keys with network users.
- The TA chooses session keys and distributes them in encrypted form upon request of network users.
- We will need to define appropriate attack models and adversarial goals.
- It is difficult to formulate precise definitions because SKDS often do not include mutual identification of Alice and Bob.
- In this section, we mainly give a historical tour of some SKDS.

Protocol: *Needham-Schroeder SKDS (1978)*

1. Alice chooses a random number, r_A. Alice sends $ID(Alice)$, $ID(Bob)$ and r_A to the TA.
2. The TA chooses a random **session key**, K. Then it computes $t_{Bob} = e_{K_{Bob}}(K \parallel ID(Alice))$ (a **ticket** to Bob) and $y_1 = e_{K_{Alice}}(r_A \parallel ID(Bob) \parallel K \parallel t_{Bob})$ and sends y_1 to Alice.
3. Alice decrypts y_1 using her key K_{Alice}, obtaining K and t_{Bob}. Then Alice sends t_{Bob} to Bob.
4. Bob decrypts t_{Bob} using his key K_{Bob}, obtaining K. Then, Bob chooses a random number r_B and computes $y_2 = e_K(r_B)$. Bob sends y_2 to Alice.
5. Alice decrypts y_2 using the session key K, obtaining r_B. Then Alice computes $y_3 = e_K(r_B - 1)$ and sends y_3 to Bob.

Validity Checks in the NS Protocol

- in this protocol, the term **validity checks** refers to checking that decrypted data has correct format and contains expected information (note that there are no MACs being used in the *Needham-Schroeder SKDS*)
- when Alice decrypts y_1, she checks to see that the plaintext $d_{K_{Alice}}(y_1) = r_A \parallel ID(Bob) \parallel K \parallel t_{Bob}$ for some K and t_{Bob}
- if the above condition holds, then Alice "accepts", otherwise Alice "rejects"
- when Bob decrypts y_3, he checks to see that the plaintext $d_K(y_3) = r_B - 1$
- if the above condition holds, then Bob "accepts", otherwise Bob "rejects"

The Needham-Schroeder SKDS (diagram)

TA | Alice | Bob

$$\xleftarrow{\quad Alice, Bob, r_A \quad}$$

$$t_{Bob} = e_{K_{Bob}}(K \parallel Alice)$$

$$\xrightarrow{e_{K_{Alice}}(r_A \parallel Bob \parallel K \parallel t_{Bob})}$$

$$\xrightarrow{\quad t_{Bob} \quad}$$

$$\xleftarrow{\quad e_K(r_B) \quad}$$

$$\xrightarrow{e_K(r_B - 1)}$$

Denning-Sacco Known Session-key Attack (1981)

- suppose Oscar records a session of the *Needham-Schroeder SKDS* between Alice and Bob, and then he somehow obtains the session key, K
- Oscar can then initiate a new session of the *Needham-Schroeder SKDS* by sending the previously used ticket, t_{Bob}, to Bob:

Oscar		Bob
	$\xrightarrow{t_{Bob} = e_{K_{Bob}}(K \parallel Alice)}$	
	$\xleftarrow{e_K(r'_B)}$	
	$\xrightarrow{e_K(r'_B - 1)}$	

Consequences of the Denning-Sacco Attack

- Bob thinks he has a "new" session key, K, shared with Alice
- this "new" key K is known to Oscar, but it might not be known to Alice (she may have thrown away the key K after the previous session terminated)
- in any reasonable formulation of adversarial goals, this would be considered a successful attack!

Protocol: *Simplified Kerberos V5 (1989)*

1. Alice chooses a random number, r_A. Alice sends $ID(Alice)$, $ID(Bob)$ and r_A to the TA.
2. The TA chooses a random session key K and a validity period (or **lifetime**), L. Then it computes $t_{Bob} = e_{K_{Bob}}(K \parallel ID(Alice) \parallel L)$ and $y_1 = e_{K_{Alice}}(r_A \parallel ID(Bob) \parallel K \parallel L)$ and sends t_{Bob} and y_1 to Alice.
3. Alice decrypts y_1 using her key K_{Alice}, obtaining K. Then Alice determines the current time, $time_A$, she computes $y_2 = e_K(ID(Alice) \parallel time_A)$ and she sends t_{Bob} and y_2 to Bob.
4. Bob decrypts t_{Bob} using his key K_{Bob}, obtaining K. Then, Bob computes $y_3 = e_K(time_A)$. Bob sends y_3 to Alice.
5. Alice decrypts y_3 using the session key K, and verifies that $time_A$ is the result.

Validity Checks in Kerberos

- when Alice decrypts y_1, she checks to see that the plaintext $d_{K_{Alice}}(y_1) = r_A \parallel ID(Bob) \parallel K \parallel L$ for some K and L
- if the above condition does not hold, then Alice "rejects" and aborts the current session
- when Bob decrypts y_2 and t_{Bob}, he checks to see that the plaintext $d_K(y_3) = ID(Alice) \parallel time_A$ and the plaintext $d_{K_{Bob}}(t_{Bob}) = K \parallel ID(Alice) \parallel L$, where $ID(Alice)$ is the same in both plaintexts and $time_A \in L$
- if the above conditions hold, then Bob "accepts", otherwise Bob "rejects"
- when Alice decrypts y_3, she checks that $d_K(y_3) = time_A$
- if the above condition holds, then Alice "accepts", otherwise Alice "rejects"

Kerberos V5 (diagram)

TA	Alice	Bob
$\xleftarrow{\quad Alice, Bob, r_A \quad}$		
$\xrightarrow{\quad t_{Bob}, y_1 \quad}$		
	$\xrightarrow{\quad t_{Bob}, y_2 \quad}$	
	$\xleftarrow{\quad e_K(time_A) \quad}$	

where

$$
\begin{aligned}
t_{Bob} &= e_{K_{Bob}}(K \parallel Alice \parallel L), \\
y_1 &= e_{K_{Alice}}(r_A \parallel Bob \parallel K \parallel L), \quad \text{and} \\
y_2 &= e_K(Alice \parallel time_A)
\end{aligned}
$$

Comments on Needham-Schroeder and Kerberos

- Steps 1–3 of *Needham-Schroeder* comprise key distribution; steps 4 and 5 provide **key confirmation** from Alice to Bob. In *Kerberos*, **mutual key confirmation** is accomplished in steps 3 and 4.
- In *Needham-Schroeder*, information intended for Bob is doubly encrypted, which adds unnecessary complexity to the protocol; this double encryption was removed in *Kerberos*.
- Partial protection against the Denning-Sacco replay attack is provided in *Kerberos* by the authenticated timestamp, y_2. However:
  - timestamps require reliable, synchronized clocks; and
  - protocols using timestamps are hard to analyze and it is difficult to give convincing security proofs for them.

  For the above reasons, random challenges may be preferred to timestamps.

Comments on Needham-Schroeder and Kerberos (cont.)

- It is not a good idea to use the session key, K, to encrypt information in an SKDS, because information about K may be revealed; this makes it difficult to ensure semantic security of the session key.
- Key confirmation is not required for an SKDS protocol to be considered secure! Furthermore, possession of a key during the SKDS does not imply possession of the key at a later time. For these reasons, it is now generally recommended that key confirmation be **omitted** from SKDS.
- In *Needham-Schroeder* and *Kerberos*, encryption is used to provide both secrecy and authenticity — it is much better to use a MAC to provide authenticity.

Improving the Second Flow of Needham-Schroeder

It is better to remove the double encryption and use MACs for authentication:

2. The TA chooses a random session key K. Then it computes $t_{Bob} = e_{K_{Bob}}(K \parallel ID(Alice))$ and $y_1 = e_{K_{Alice}}(r_A \parallel ID(Bob) \parallel K \parallel t_{Bob})$ and sends y_1 to Alice.

... should be replaced by the following:

2′. The TA chooses a random session key K. Then it computes

$$y_1 = (e_{K_{Bob}}(K), MAC_{Bob}(ID(Alice) \parallel e_{K_{Bob}}(K)),$$

and

$$y_1' = (e_{K_{Alice}}(K), MAC_{Alice}(ID(Bob) \parallel r_A \parallel e_{K_{Alice}}(K))$$

The TA sends y_1 to Bob (possibly via Alice) and y_1' to Alice.

Further Discussion

- the revised second flow does not fix the flaw in *Needham-Schroeder* found by Denning and Sacco (also note that a similar attack can be carried out in Kerberos within the specified lifetime of the key)
- to prevent the Denning-Sacco attack, the flow structure of the protocol must be modified: a "secure" protocol should involve Bob as an active participant prior to his receiving the session key, in order to prevent these kinds of replay attacks
- the solution is to have Alice contact Bob (or vice versa) **before** sending a request to the TA
- we present a more recent, secure protocol due to Bellare and Rogaway

Protocol: *Bellare-Rogaway SKDS (1995)*

1. Alice chooses a random number, r_A, and she sends $ID(Alice)$ and r_A to Bob.
2. Bob chooses a random number, r_B, and he sends $ID(Alice)$, $ID(Bob)$, r_A and r_B to the TA.
3. The TA chooses a random session key K. Then it computes

$$y_B = (e_{K_{Bob}}(K), MAC_{Bob}(ID(Alice) \parallel ID(Bob) \parallel r_B \parallel e_{K_{Bob}}(K))$$

and

$$y_A = (e_{K_{Alice}}(K), MAC_{Alice}(ID(Bob) \parallel ID(Alice) \parallel r_A \parallel e_{K_{Alice}}(K))$$

The TA sends y_B to Bob and y_A to Alice.

4. Alice decrypts y_A using her key K_{Alice}, obtaining K. Bob decrypts y_B using his key K_{Bob}, obtaining K.

Comments on the Bellare-Rogaway Protocol

- (honest) Alice and Bob "accept" if the MACs they receive are valid, and the random challenges (r_A and r_B, respectively) in the MACs are the same as the ones that they chose in steps 1 and 2 of the protocol
- the TA is authenticated to Alice and Bob by the use of MACs in step 3
- the protocol does not include authentication of Alice to Bob, or of Bob to Alice, or of Alice or Bob to the TA
- the protocol also does not provide key confirmation (from Alice to Bob, or from Bob to Alice)
- however, the protocol is a **secure (unauthenticated) key distribution protocol**, which means that it satisfies certain security properties in a known session key attack, which will be discussed later ...

Bellare-Rogaway SKDS (diagram)

TA	Bob	Alice
	$\xleftarrow{\quad Alice, r_A \quad}$	
$\xleftarrow{\quad Alice, Bob, r_A, r_B \quad}$		
$\xrightarrow{\quad y_A, y_B \quad}$		
		$\xrightarrow{\quad y_A \quad}$

where

$$
\begin{aligned}
y_B &= (e_{K_{Bob}}(K), MAC_{Bob}(ID(Alice) \parallel ID(Bob) \parallel r_B \parallel e_{K_{Bob}}(K)) \\
y_A &= (e_{K_{Alice}}(K), MAC_{Alice}(ID(Bob) \parallel ID(Alice) \parallel r_A \parallel e_{K_{Alice}}(K))
\end{aligned}
$$

Security Properties of the Bellare-Rogaway Protocol

1. If Alice, Bob and the TA are honest and the adversary is passive, then Alice and Bob compute the same session key, K, and the adversary cannot compute any information about K.
2. If an (active) adversary changes messages originating from (or being sent to) an honest participant, then that participant will "reject" with high probability.
3. If Alice and the TA are honest, and an (active) adversary impersonates Bob, then then the adversary cannot compute the session key.
4. If Bob and the TA are honest, and an (active) adversary impersonates Alice, then then the adversary cannot compute the session key.

Analysis and Discussion

- properties (1), (3) and (4) are guaranteed by using a secure encryption scheme
- proving property (2) requires analyzing the probability that the adversary can guess a correct new MAC or reuse a MAC from a previous session (similar to the analysis of the identification protocols considered earlier)
- the scheme is not secure against a known LL-key attack
- the *Bellare-Rogaway* protocol does "less" than *Needham-Schroeder* or *Kerberos*, but its simplicity means that it can be analyzed rigourously

Consequences of the Security Properties

- if (honest) Alice accepts, then she has received a session key, K, and she can be confident that Bob and the TA are the only parties that can determine K
- the adversary's goal is to cause an (honest) participant to accept in a situation where someone other than the (honest) intended partner of that participant knows the value of the session key K
- we argue that the adversary cannot achieve his goal, as follows:
  - if (honest) Alice accepts, then the adversary did not tamper with messages sent to or from Alice
  - hence, y_A was constructed by the TA, and K is encrypted only with the keys K_{Alice} and K_{Bob}
  - therefore, the adversary will not learn the value of K if Bob and the TA are honest

Protocol: *Diffie-Hellman key agreement*

1. U chooses a_U at random, where $0 \leq a_U \leq q-1$. Then U computes

$$b_U = \alpha^{a_U} \bmod p$$

   and sends b_U to V.

2. V chooses a_V at random, where $0 \leq a_V \leq q-1$. Then V computes

$$b_V = \alpha^{a_V} \bmod p$$

   and sends b_V to U.

3. U computes

$$K = (b_V)^{a_U} \bmod p$$

   and V computes

$$K = (b_U)^{a_V} \bmod p.$$

Security wrt Passive and Active Attacks

- the key $K = \textbf{CDH}(\alpha, b_U, b_V)$
- assuming that **DDH** is intractible, a passive adversary cannot compute any information about K
- the scheme is insecure against active attacks, however

Diffie-Hellman Key Agreement is supposed to look like this:

$$U \qquad\qquad\qquad\qquad V$$

$$\xrightarrow{\quad \alpha^{a_U} \quad}$$

$$\xleftarrow{\quad \alpha^{a_V} \quad}$$

Intruder-in-the-middle Attack

In an intruder-in-the-middle attack, W will intercept messages between U and V and substitute his own messages:

$$
\begin{array}{ccccc}
U & & W & & V \\
 & \xrightarrow{\alpha^{a_U}} & & \xrightarrow{\alpha^{a'_U}} & \\
 & \xleftarrow{\alpha^{a'_V}} & & \xleftarrow{\alpha^{a_V}} &
\end{array}
$$

At the end of the protocol, U has actually established the secret key $\alpha^{a_U a'_V}$ with W, and V has established a secret key $\alpha^{a'_U a_V}$ with W.

Secure Authenticated Key Agreement Protocols

Here is a design strategy to obtain a secure authenticated key agreement protocol:

1. start with a secure mutual identification protocol
2. replace the random challenges of U and V, say r_U and r_V respectively, by b_U and b_V respectively, where $b_U = \alpha^{a_U} \bmod p$ and $b_V = \alpha^{a_V} \bmod p$
3. then both U and V can compute the session key $K = \mathbf{CDH}(\alpha, b_U, b_V)$, but the adversary cannot do so

Protocol: *Public-key Mutual Identification Protocol*

1. U chooses a random challenge r_U and sends it to V.
2. V chooses a random challenge r_V. Then he computes

$$y_V = sig_V(ID(U) \parallel r_V \parallel r_U).$$

   and sends $(\mathbf{Cert}(V), r_V, y_V)$ to U.
3. U verifies y_V using ver_V (and she verifies $\mathbf{Cert}(V)$ using ver_{TA}). If the signature y_V is not valid, then she outputs "reject" and quits. Otherwise, she outputs "accept", computes

$$y_U = sig_U(ID(V) \parallel r_U \parallel r_V),$$

   and sends $(\mathbf{Cert}(U), y_U)$ to V.
4. V verifies y_U using ver_U (and he verifies $\mathbf{Cert}(U)$ using ver_{TA}). If the signature y_U is not valid, then U outputs "reject"; otherwise, he outputs "accept".

We now transform the previous protocol into a key agreement scheme. This is basically a simplified version of the *Station-to-station* (or *STS*) protocol.

Protocol: *(Simplified) station-to-station protocol*

1. U chooses a random number a_U, $0 \leq a_U \leq q-1$. Then she computes

$$b_U = \alpha^{a_U} \bmod p$$

   and sends it to V.

2. V chooses a random number a_V, $0 \leq a_V \leq q-1$. Then he computes

$$\begin{aligned} b_V &= \alpha^{a_V} \bmod p \\ K &= (b_U)^{a_V} \bmod p, \quad \text{and} \\ y_V &= \mathit{sig}_V(\mathit{ID}(U) \parallel \alpha^{a_V} \parallel \alpha^{a_U}). \end{aligned}$$

   Then V sends $(\mathbf{Cert}(V), b_V, y_V)$ to U.

Protocol: *(Simplified) station-to-station protocol (cont.)*

3. U verifies y_V using ver_V (and she verifies $\mathbf{Cert}(V)$ using ver_{TA}). If the signature y_V is not valid, then she outputs "reject" and quits. Otherwise, she outputs "accept", computes

$$\begin{aligned} K &= (b_V)^{a_U} \bmod p, \quad \text{and} \\ y_U &= sig_U(ID(V) \parallel \alpha^{a_U} \parallel \alpha^{a_V}), \end{aligned}$$

   and sends $(\mathbf{Cert}(U), y_U)$ to V.

4. V verifies y_U using ver_U (and he verifies $\mathbf{Cert}(U)$ using ver_{TA}). If the signature y_U is not valid, then U outputs "reject"; otherwise, he outputs "accept".

Station-to-Station Protocol

U		V
$b_U = \alpha^{a_U}$	$\xrightarrow{b_U}$	
	$\xleftarrow{b_V, y_V}$	$b_V = \alpha^{a_V}$ $y_V = sig_V(U \parallel b_V \parallel b_U)$
$y_U = sig_U(V \parallel b_U \parallel b_V)$	$\xrightarrow{y_U}$	

Security Properties of STS

- *STS* is secure against known session key attacks and provides perfect forward secrecy
- *STS* is a secure mutual identification scheme (i.e., if the adversary is active during a given flow of the protocol, then no honest participant will "accept" after that time)
- in addition, the scheme is a secure KAS wrt a passive adversary (i.e., U and V can both compute the same session key, K, and the adversary cannot compute any information about K)
- if U "accepts", it means that she is confident that
  - she has been communicating with V
  - U and V can compute the same session key, and
  - no one other than V can compute any information about the session key

Key Authentication and Key Confirmation

Suppose U and V are honest, and they execute an SKDS or KAS. At the end of the session, U and V should each be able to compute the same session key, K, whose value should be unknown to the adversary. Suppose that U "accepts". The following properties discuss various types of assurance that may be provided to U:

implicit key authentication
: U is assured that no one other than V can compute K

implicit key confirmation
: U is assured that V **has the necessary information** in order to be able to compute K, and no one other than V can compute K

explicit key confirmation
: U is assured that V **actually has computed** K, and no one other than V can compute K

Key Authentication/Confirmation in Specific Protocols

- the *Needham-Schroeder* and *Kerberos* SKDS provide explicit key confirmation
- the *Station-to-Station* KAS provides implicit key confirmation
- the *Bellare-Rogaway* SKDS provides implicit key authentication
- it is now generally accepted that explicit key confirmation is not an important requirement of an SKDS or KAS
- if desired, a protocol to ensure explicit key confirmation can be carried out at some later time, e.g., when the session key is going to be used for its intended purpose

Identification and Key Authentication/Confirmation

- implicit or explicit key confirmation is not possible without simultaneous mutual identification (entity authentication)
- implicit key authentication is possible without identification
- of course, mutual identification can be done in the absence of any key distribution or key agreement
- a KAS (or SKDS) with key confirmation is "secure" if it is a secure mutual identification scheme, and the specified type of key confirmation is acheived

Matsumoto-Takashima-Imai Protocols

- the *MTI* protocols are public-key key agreement schemes that use LL-keys (authenticated via certificates)
- the algebraic setting is the same as for *Diffie-Hellman* key agreement schemes, i.e., we work in a subgroup of ${\mathbb{Z}_p}^*$ of prime order q, in which the **DDH** problem is believed to be intractable
- the protocols are two-pass key agreement protocols **without** identification, in which signatures are **not** used
- every user U has a private LL-key a_U and a corresponding public key

$$b_U = \alpha^{a_U} \bmod p$$

- the users' public keys are signed by the TA and stored on certificates, as usual

Protocol: *MTI/A0 key agreement*

1. U chooses r_U at random, where $0 \leq r_U \leq q-1$. Then U computes

$$s_U = \alpha^{r_U} \bmod p$$

   and sends $\mathbf{Cert}(U)$ and s_U to V.

2. V chooses r_V at random, where $0 \leq r_V \leq q-1$. Then V computes

$$s_V = \alpha^{r_V} \bmod p$$

   and sends $\mathbf{Cert}(V)$ and s_V to U.

3. U computes

$$K = (b_V)^{r_U}(s_V)^{a_U} \bmod p$$

   and V computes

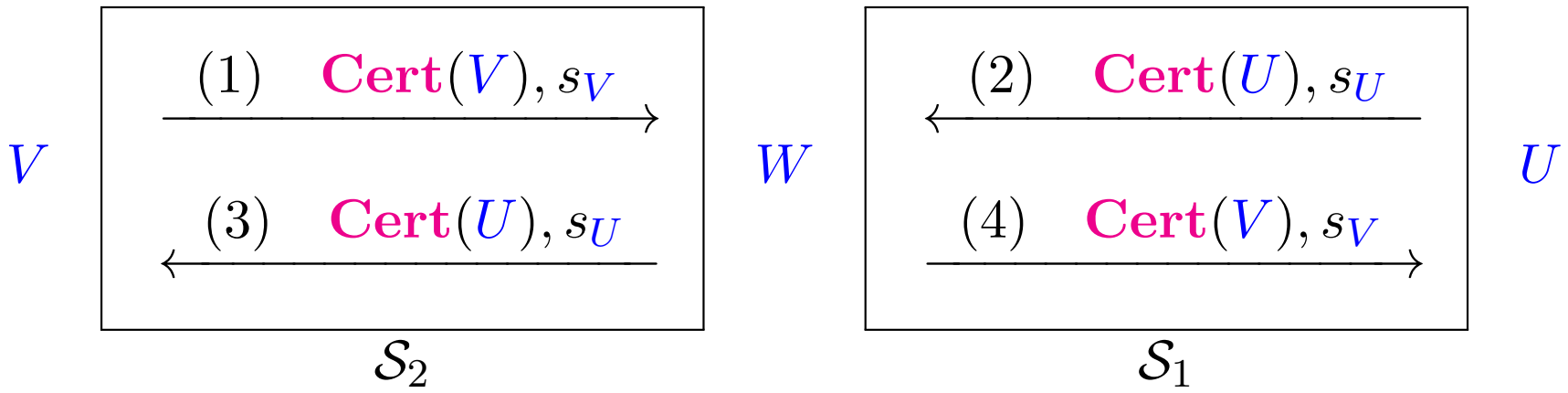
$$K = (b_U)^{r_V}(s_U)^{a_V} \bmod p.$$

Analysis of MTI/A0

- the session key is
$$K = \alpha^{r_U a_V + r_V a_U} \bmod p$$
- the *MTI/A0* protocol attempts to provide implicit key authentication without identification (similar to the *Bellare-Rogaway* SKDS)
- the protocol is secure against passive attacks (assuming the **DDH** problem is intractable)
- the protocol is apparently secure against intruder-in-the-middle attacks
- however, the *MTI/A0* protocol is vulnerable to a certain kind of parallel session known session key attack, as well as the more innovative **Burmester triangle attack**

A Parallel Session Attack

- W pretends to be V in a session $\mathcal{S}_1$ with U; and W pretends to be U in a session $\mathcal{S}_2$ with V:



- then W requests the key K for session $\mathcal{S}_2$, which is allowed in a known session key attack
- K is also the key for session $\mathcal{S}_1$

The Triangle Attack

V	W	U
	$\xleftarrow{\mathbf{Cert}(U),\, s_U}$	
	$\xrightarrow{\mathbf{Cert}(V),\, s_V}$	
$\xleftarrow{\mathbf{Cert}(W),\, s_U}$		
$\xrightarrow{\mathbf{Cert}(V),\, s'_V}$		
		$\xrightarrow{\mathbf{Cert}(W),\, s_V}$
		$\xleftarrow{\mathbf{Cert}(U),\, s'_U}$

Analysis of the Triangle Attack

- the session keys for the three sessions are as follows:

$$\begin{aligned} K_1 &= \alpha^{r_U a_V + r_V a_U} \\ K_2 &= \alpha^{r_U a_V + r'_V a_W} \\ K_3 &= \alpha^{r'_U a_W + r_V a_U} \end{aligned}$$

- therefore

$$K_1 = \frac{K_2 K_3}{(s'_V s'_U)^{a_W}}$$

- suppose that W requests K_2 and K_3 (in a known session key attack)
- then W can compute K_1

A Secure (?) Variation

- the parallel session attack can be carried out because the key is a symmetric function of the inputs provided by the two parties (i.e., $K((r_U, a_U), (r_V, a_V)) = K((r_V, a_V), (r_U, a_U)))$
- to eliminate the attack, we should destroy this symmetry property
- the triangle attack can be eliminated through the use of a **key derivation function**, which is a "random" public function used to derive the session key from the values $\alpha^{r_U a_V}$ and $\alpha^{r_V a_U}$
- suppose that the key derivation function is defined as

$$K = h_{\mathbf{RO}}(\alpha^{r_U a_V} \parallel \alpha^{r_V a_U}),$$

  where $h_{\mathbf{RO}}$ is a truly random function (i.e., a **random oracle**)
- the resulting scheme is conjectured to be secure by Blake-Wilson, Johnson and Menezes

Further Settings For Two-Party Key Establishment Protocols

- we have considered two-party key agreement schemes in the public-key setting
- it is also possible to study **key transport schemes**, where one party chooses the session key and "transports" it to the other party (i.e., by encrypting it)
- KAS and KTS can also be studied in the secret-key setting, where signature schemes are replaced by MACs and public-key encryption is replaced by secret-key encryption
- in the public-key setting, **identity-based schemes** have also been devised, in which public keys are derived from users' identities; this eliminates the need for certificates to authenticate public keys

Table of Contents

Multicast Security

- a **multicast message** is a message that has many designated receivers, i.e., one-to-many communication as opposed to one-to-one communication
- a **multicast network** is a network of users in which it is possible to send messages simultanously to all of the users
- two "benchmark scenarios" for multicast security:

single source broadcast
: a single entity broadcasts information to a long-lived, possibly dynamic, **group** (consisting of some or all of the users in the network)

virtual conference
: a subset of the network users wish to form a small virtual conference, i.e., a short-lived, static, group

Possible Goals

- in a single source broadcast, we might want to achieve one or more of the following goals:
  - confidentiality, data integrity
  - key revocation, key updating
  - source authentication
  - copyright protection
- in a virtual conference, we might want to achieve one or more of the following goals:
  - establishment of a group session key (which enables confidentiality and data integrity within the group)
  - source or group authentication
  - anonymity
  - shared access control

Cryptographic Tools

New cryptographic tools are required to enable multicast security:

secret sharing schemes

distributing shares of a "secret" in such a way that a specified threshold of shares is required to reconstruct the secret; used as a building block in more complicated protocols

group key predistribution schemes

KPS in which keys are associated with groups of users of a certain size; secure against coalitions of a certain size

broadcast encryption/exclusion

broadcasting an encrypted message in such a way that only users in a specified group can decrypt the message; secure against coalitions of a certain size; used for group session key distribution and key updating

Cryptographic Tools (cont.)

tracing schemes

also called fingerprinting schemes; they comprise techniques and algorithms to facilitate tracing of illegally redistributed keys and/or content back to the owners

interactive group KAS

KAS in which a session key is established for a specified group of users

multireceiver authentication codes

a MAC in which all the users in a specified group can validate the authenticity and/or the source of data transmitted by one of the group members

Secret Sharing (overview)

- various types of shared control schemes depend on a cryptographic primitive called a (t, n)-**threshold scheme**
- let t and n be positive integers, $t \leq n$
- we have a trusted authority, denoted TA, and n users, denoted $U_1, \ldots, U_n$
- the TA has a secret value $K \in \mathcal{K}$, called a **secret** or a **key**, where $\mathcal{K}$ is a specified finite set

Secret Sharing (overview)

- the TA uses a **share generation algorithm** to split K into n **shares**, denoted $s_1, \ldots, s_n$
- each share $s_i \in \mathcal{S}$, where $\mathcal{S}$ is a specified finite set
- for every i, $1 \leq i \leq n$, the share s_i is transmitted by the TA to user U_i using a secure channel
- the following two properties should hold:
  1. a **reconstruction algorithm** can be used to reconstruct the secret, given any t of the n shares,
  2. no $t-1$ shares reveal any information as to the value of the secret

An (n, n)-Threshold Scheme

- suppose $K \in \mathbb{Z}_m$ is the secret
- let $s_1, \ldots, s_{n-1}$ be chosen independently and uniformly at random from $\mathbb{Z}_m$
- let

$$s_n = K - \sum_{i=1}^{n-1} s_i \bmod m$$

- $s_1, \ldots, s_n$ are shares of an (n, n)-threshold scheme:
  1. $K = \sum s_i \bmod m$
  2. given all the shares except s_j, K could take on any value, depending on the value of the share s_j

Shamir Threshold Scheme

- in 1979, Shamir showed how to construct a (t, n)-threshold scheme based on polynomial interpolation over $\mathbb{Z}_p$, where p is prime
- let $p \geq n + 1$ be a prime
- let $\mathcal{K} = \mathcal{S} = \mathbb{Z}_p$
- in an initialization phase, $x_1, x_2, \ldots, x_n$ are defined to be n distinct non-zero elements of $\mathbb{Z}_p$
- the TA gives x_i to U_i, for all i, $1 \leq i \leq n$
- the x_i's are public information

Share Generation for the Shamir Scheme

- given a secret $K \in \mathbb{Z}_p$, the share generation algorithm is as follows:
  1. the TA chooses $a_1, \ldots, a_{t-1}$ independently and uniformly at random from $\mathbb{Z}_p$
  2. the TA defines
  $$a(x) = K + \sum_{j=1}^{t-1} a_j \, x^j$$
  (note that $a(x) \in \mathbb{Z}_p[x]$ is a random polynomial of degree at most $t-1$, such that the constant term is the secret, K)
  3. for $1 \leq i \leq n$, the TA constructs the share $s_i = a(x_i)$

Reconstruction Algorithm for the Shamir Scheme

- suppose users $U_{i_1}, \ldots, U_{i_t}$ want to determine K
- they know that $s_{i_j} = a(x_{i_j})$, $1 \leq j \leq t$
- since $a(x)$ is a polynomial of degree at most $t-1$, they can determine $a(x)$ by Lagrange interpolation; then $K = a(0)$
- recall the Lagrange interpolation formula:

$$a(x) = \sum_{j=1}^{t} s_{i_j} \prod_{1 \leq k \leq t, k \neq j} \frac{x - x_{i_k}}{x_{i_j} - x_{i_k}}$$

- set $x = 0$:

$$K = \sum_{j=1}^{t} s_{i_j} \prod_{1 \leq k \leq t, k \neq j} \frac{-x_{i_k}}{x_{i_j} - x_{i_k}} = \sum_{j=1}^{t} s_{i_j} \prod_{1 \leq k \leq t, k \neq j} \frac{x_{i_k}}{x_{i_k} - x_{i_j}}$$

Reconstruction Algorithm for the Shamir Scheme (cont.)

- for $1 \leq j \leq t$, define

$$b_j = \prod_{1 \leq k \leq t, k \neq j} \frac{x_{i_k}}{x_{i_k} - x_{i_j}}$$

- the b_j's do not depend on the shares, so they can be precomputed (for a given subset of t users)
- then K can be computed from the formula

$$K = \sum_{j=1}^{t} b_j \, s_{i_j}$$

Example

- suppose that $p = 17$, $t = 3$, and $n = 5$; and the public x-co-ordinates are $x_i = i$, $1 \leq i \leq 5$
- suppose that the group $G = \{U_1, U_3, U_5\}$ wishes to compute K, given their shares $8, 10$, and 11, respectively
- it is possible to (pre)compute b_1, b_2, and b_3:

$$\begin{aligned} b_1 &= \frac{x_3 x_5}{(x_3 - x_1)(x_5 - x_1)} \bmod 17 \\ &= 3 \times 5 \times (-2)^{-1} \times (-4)^{-1} \bmod 17 \\ &= 4, \\ b_2 &= 3, \quad \text{and} \\ b_3 &= 11 \end{aligned}$$

- then the users in G can (collectively) compute K as follows:

$$K = 4 \times 8 + 3 \times 10 + 11 \times 11 \bmod 17 = 13$$

Security of the Shamir Scheme

- suppose users $U_{i_1}, \ldots, U_{i_{t-1}}$ want to determine K
- they know that $s_{i_j} = a(x_{i_j})$, $1 \leq j \leq t-1$
- let K_0 be arbitrary
- by Lagrange interpolation, there is a unique polynomial $a_0(x)$ such that $s_{i_j} = a_0(x_{i_j})$ for $1 \leq j \leq t-1$ and such that $K_0 = a_0(0)$
- hence no value of K can be ruled out, given the shares held by $t-1$ users

Key Distribution Patterns

- suppose we have a TA and a network of n users, $\mathcal{U} = \{U_1, \ldots, U_n\}$
- the TA chooses v random keys, say $k_1, \ldots, k_v \in \mathcal{K}$, where $(\mathcal{K}, +)$ is an additive abelian group, and gives a (different) subset of keys to each user
- a **key distribution pattern** is a public v by n incidence matrix, denoted M, which has entries in $\{0, 1\}$
- M specifies which users are to receive which keys: user U_j is given the key k_i if and only if $M[i, j] = 1$
- for a subset $P \subseteq \mathcal{U}$, define

$$\mathit{keys}(P) = \{i : M[i, j] = 1 \text{ for all } U_j \in P\}$$

Key Distribution Patterns (cont.)

- observe that

$$keys(P) = \bigcap_{U_j \in P} keys(U_j)$$

- if $keys(P) \neq \emptyset$, then the **group key** k_P for the set P is defined to be

$$k_P = \sum_{i \in keys(P)} k_i$$

- each member of P can compute k_P (no interaction requred)
- suppose $F \cap P = \emptyset$
- the coalition F can compute k_P if and only if

$$keys(P) \subseteq \bigcup_{U_j \in F} keys(U_j)$$

A Trivial Example

Suppose $n = 4$, $v = 6$, and the matrix M is as follows:

$$M = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$keys(U_1) = \{1, 2, 3\}$, $keys(U_2) = \{1, 4, 5\}$, and $keys(U_1, U_2) = \{1\}$, so $k_{\{U_1,U_2\}} = k_1$.

A Trivial Generalization

- it is always possible to construct a trivial $\binom{n}{2}$ by n KDP, in which any two users have exactly one common key, and each key is given to exactly two users
- in this KDP, each user must store $n-1$ keys
- the group key for any two users is the unique key that they both possess
- there are $\binom{n}{2}$ group keys
- each group key $k_{\{U_j,U_{j'}\}}$ is secure against the coalition $\mathcal{U}\backslash\{U_j,U_{j'}\}$ having size $n-2$

A More Interesting Example

Suppose $n = 7$, $v = 7$, and the matrix M is as follows:

$$M = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

$\mathit{keys}(U_1) = \{1, 4, 6, 7\}$, $\mathit{keys}(U_2) = \{1, 2, 5, 7\}$, and $\mathit{keys}(U_1, U_2) = \{1, 7\}$, so $k_{\{U_1, U_2\}} = k_1 + k_7$.

Example (cont.)

- no other user has both k_1 and k_2, so $k_{\{U_1,U_2\}}$ is secure against any other individual user
- however, U_3 and U_4 (for example) together can compute $k_{\{U_1,U_2\}}$ by pooling their secret information
- in this example, every pair of users can compute a key that is secure against any other individual user
- this scheme enables the construction of $\binom{7}{2} = 21$ group keys for pairs of users

Fiat-Naor KDPs

- let $1 \leq w \leq n$ and define

$$v = \sum_{i=0}^{w} \binom{n}{i}$$

- the **Fiat-Naor w-KDP** is the v by n matrix whose rows are the incidence vectors of all subsets of $\mathcal{U}$ having cardinality at least $n - w$
- there is a group key for any subset $P \subseteq \mathcal{U}$, that is secure against any disjoint coalition F of size at most w
- Proof: $|F| \leq w$, so $|\mathcal{U} \backslash F| \geq n - w$. Hence, there exists a key k_i given to the users in $\mathcal{U} \backslash F$ and to no other user. $P \subseteq (\mathcal{U} \backslash F)$, so all users in P have k_i and no user in F has k_i.

An Example

Suppose $n = 6$ and $w = 1$. The Fiat-Naor 1-KDP has $v = 7$, and the matrix M is as follows:

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$keys(U_1) = \{1, 2, 3, 4, 5, 6\}$, $keys(U_3) = \{1, 2, 3, 4, 6, 7\}$, $keys(U_4) = \{1, 2, 3, 5, 6, 7\}$, and $keys(U_1, U_3, U_4) = \{1, 2, 3, 6\}$, so $k_{\{U_1,U_3,U_4\}} = k_1 + k_2 + k_3 + k_6$. No other single user can compute this key.

Cover-free Families

- a **set system** is a pair $(X, \mathcal{A})$, where X is a finite set of **points** and $\mathcal{A}$ is a set of subsets of X called **blocks**
- denote $X = \{x_1, \ldots, x_v\}$ and denote $\mathcal{A} = \{A_1, \ldots, A_n\}$
- $(X, \mathcal{A})$ is a (t, w)-**cover-free family** provided that, for any two disjoint subsets of blocks $P, F \subseteq \mathcal{A}$, where $|P| = t$ and $|F| = w$, it holds that

$$\bigcap_{A_i \in P} A_i \not\subseteq \bigcup_{A_j \in F} A_j$$

- a (t, w)-cover-free family will be denoted as a (t, w)-CFF(v, n) if $|X| = v$ and $|\mathcal{A}| = n$

Mitchell-Piper KDPs

- the **incidence matrix** of $(X, \mathcal{A})$ is the $v \times n$ matrix $M = (m_{i,j})$ in which
$$m_{i,j} = \begin{cases} 1 & \text{if } x_i \in A_j \\ 0 & \text{otherwise} \end{cases}$$
- a **Mitchell-Piper (t, w)-KDP** is a KDP in which there is a key for every group of t users, and each such key is secure against any disjoint coalition of at most w users
- the KDP given by a $v \times n$ matrix M yields a Mitchell-Piper (t, w)-KDP if and only if M is the incidence matrix of a (t, w)-CFF(v, n)
- the blocks of the CFF correspond to the columns of M (i.e., the blocks are *keys*$(U_1), \ldots,$ *keys*(U_n))

A $(2,1)$-CFF$(7,7)$

$$
\begin{aligned}
M &= \begin{pmatrix}
1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 & 1
\end{pmatrix} \\
X &= \{1,\ldots,7\}, \quad \text{and} \\
\mathcal{A} &= \{\{1,4,6,7\},\{1,2,5,7\},\{1,2,3,6\},\{2,3,4,7\}, \\
&\quad \{1,3,4,5\},\{2,4,5,6\},\{3,5,6,7\}\}
\end{aligned}
$$

Non-constructive Existence Results

- given t, w and n, we want to construct (t, w)-CFF(v, n) for v as small as possible
- let M be a v by n matrix, having entries from $\{0, 1\}$, whose columns are labelled $1, \ldots, n$
- let $0 < \rho < 1$, and suppose that every entry of M is (independently) defined to be a "1" with probability ρ
- the probability that the matrix M constructed in this fashion is a (t, w)-CFF(v, n) will be shown to be positive, provided that certain numerical conditions on the parameters are satisfied
- this will prove (non-constructively) that a (t, w)-CFF(v, n) exists for certain values of t, w, v and n

Non-constructive Existence Results (cont.)

- let the matrix M be given
- suppose that $P, F \subseteq \{1, \ldots, n\}$, $|P| = t$, $|F| = w$ and $P \cap F = \emptyset$
- then, a given row i of M satisfies the property (*) provided that the entries in the columns in P are all "1"s and the entries in the columns in F are all "0"s
- define a random variable

$$X(P, F) = \begin{cases} 0 & \text{if there exists a row } i \text{ such that (*) is satisfied} \\ 1 & \text{otherwise} \end{cases}$$

- **First Observation:** if $X(P, F) = 1$, then M is not the incidence matrix of a (t, w)-CFF(v, n)

Non-constructive Existence Results (cont.)

- the probability that (*) is satisfied with respect to a given row is $\rho^t(1-\rho)^w$
- the probability that (*) is not satisfied with respect to all v rows is $\left(1-\rho^t(1-\rho)^w\right)^v$
- the expected value of $X(P,F)$ is

$$\begin{aligned}\mathsf{Exp}[X(P,F)] &= \mathsf{Pr}[X(P,F)=0]\times 0 + \mathsf{Pr}[X(P,F)=1]\times 1\\ &= \mathsf{Pr}[X(P,F)=1]\\ &= \left(1-\rho^t(1-\rho)^w\right)^v\end{aligned}$$

- in order to minimize $\mathsf{Exp}[X(P,F)]$, we should maximize $\rho^t(1-\rho)^w$
- elementary calculus shows that we should define $\rho = t/(t+w)$

Non-constructive Existence Results (cont.)

- define the random variable

$$X = \sum_{\{(P,F):|P|=t,|F|=w,P\cap F=\emptyset\}} X(P,F)$$

- **Second Observation:** $X = 0$ if and only if M is the incidence matrix of a (t,w)-CFF(v,n)
- it is easy to see that

$$\begin{aligned}
\mathsf{Exp}[X] &= \sum_{\{(P,F):|P|=t,|F|=w,P\cap F=\emptyset\}} \mathsf{Exp}[X(P,F)] \\
&= \binom{n}{t}\binom{n-t}{w}\left(1-\rho^t(1-\rho)^w\right)^v \\
&< n^{t+w}\left(1-\frac{t^t w^w}{(t+w)^{t+w}}\right)^v
\end{aligned}$$

Non-constructive Existence Results (cont.)

- if Exp$[X] < 1$, then there is a (t, w)-CFF(v, n)
- define

$$p_{t,w} = 1 - \frac{t^t w^w}{(t+w)^{t+w}}$$

- Exp$[X] < 1$ if

$$\begin{aligned} n^{t+w}(p_{t,w})^v &< 1 \quad \Leftrightarrow \\ (t+w)\log_2 n + v\log_2(p_{t,w}) &< 0 \quad \Leftrightarrow \\ v &> \frac{(t+w)\log_2 n}{-\log_2(p_{t,w})} \end{aligned}$$

- in other words, there exist (t, w)-CFF(v, n) in which v is $O(\log n)$ (for any fixed t and w)

Example

- suppose $t = 2$ and $w = 1$
- then

$$\begin{aligned} \rho &= 2/3 \\ p_{2,1} &= 1 - \frac{2^2 1^1}{3^3} = \frac{23}{27}, \quad \text{and} \\ \frac{2+1}{-\log_2 p_{2,1}} &\approx 12.97 \end{aligned}$$

- for all $n \geq 2$, there exists a $(1,2)$-CFF(v,n) in which $v \leq 13 \log_2 n$

Non-constructive Existence Results (summary)

- define

$$\rho = \frac{t}{t+w}$$

  and

$$p_{t,w} = 1 - \frac{t^t w^w}{(t+w)^{t+w}}$$

- recall that Exp$[X] < n^{t+w}(p_{t,w})^v$
- if

$$v > \frac{(t+w)\log_2 n}{-\log_2(p_{t,w})}$$

  then Exp$[X] < 1$, and hence a (t,w)-CFF(v,n) exists
- this is an existence proof, so we do not have an explicit formula or an efficient deterministic algorithm to construct the CFF
- we will consider two possible approaches to making this method practical:

Error Probabilities

1. generate a random M (as described in the existence proof), and then check to see if it is a CFF
2. choose v such that a random M will be a CFF with high probability, and don't check it

- what is the probability that a random M is not a CFF?
- we have that

$$\begin{aligned}
\mathsf{Exp}[X] &= \sum_{i=0}^{\infty}(\mathsf{Pr}[X=i] \times i) \\
&\geq \sum_{i=1}^{\infty}\mathsf{Pr}[X=i] \\
&= \mathsf{Pr}[X>0] \\
&= \mathsf{Pr}[M \text{ is not a CFF}]
\end{aligned}$$

A Practical Algorithm?

- let $s \geq 1$ (s is an error probability parameter)
- it is easy to determine v so that Exp$[X] \leq 2^{-s}$
- if
$$v > \frac{(t+w)\log_2 n + s}{-\log_2(p_{t,w})}$$
then Exp$[X] < 2^{-s}$
- then a randomly constructed M will be a (t,w)-CFF(v,n) with probability at least $1 - 2^{-s}$
- if $s = 1$, then it will take (on average) two tries until a CFF is constructed
- if $s = 20$, then almost surely a random M will be a CFF (and hence we might not even bother to verify that M is indeed a CFF)

Verification

- how do we verify that a given M is a (t, w)-CFF(v, n)?
- if M is constructed "randomly", it is necessary to check all possible (P, F) to see that the CFF property is satisfied
- the complexity of this algorithm is $O\left(\binom{n}{t}\binom{n-t}{w}(t+w)v\right)$
- for fixed t, w, this simpifies to $O\left(n^{t+w}\log n\right)$
- verification is reasonably fast if $t + w$ is small, or if n is small

Applications of (t, w)-CFF to Multicast Security

group key predistribution

A (t, w)-CFF can be used to construct a KDP in which predefined keys for groups of size at most t exist, that are secure against coalitions of size at most w.

group session key distribution

The TA can use an existing group key (from the above-described KDP) to encrypt a group session key, which is then broadcasted to the group.

blacklisting (broadcast exclusion)

A $(1, w)$-CFF allows the TA to broadcast a group (session) key, K, to everyone in the network **except** for a specified set of at most w excluded users.

This technique can be used for key revocation in a network.

Applications of (t, w)-CFF to Multicast Security

anti-jamming

A $(1, w)$-CFF allows a TA to broadcast a message using a set of carrier fequencies so that no coalition of at most w users can "jam" the communication to any intended receiver.

network source authentication

A $(1, w)$-CFF enables everyone in a network to verify that a given message was transmitted by the TA (and not by someone else); the scheme is secure against coalitions of size at most w.

Blacklisting

- for every

$$i \notin \bigcup_{U_j \in F} \mathit{keys}(U_j),$$

  the TA computes $y_i = e_{k_i}(K)$ using a secure secret-key encryption scheme
- the following properties hold:
  1. no user $U_j \in F$ can compute K (even if the users in F combine all their information)
  2. if $|F| \leq w$, then every user $U_h \notin F$ can decrypt K, because

$$\mathit{keys}(U_h) \not\subseteq \bigcup_{U_j \in F} \mathit{keys}(U_j)$$

  (this is the $(1, w)$ cover-free property).

Anti-jamming

- a $(1, w)$-CFF is used to distribute **carrier frequencies** to the network users
- every user knows a subset of the v carrier frequencies
- the TA broadcasts v copies of the same message, using all v carrier frequencies
- a bad user can "jam" a frequency, destroying the ability of that frequency to transmit a message
- a coalition of bad users, F, can jam all the frequencies that they know jointly
- if $|F| \leq w$, then every user not in F still receives at least one copy of the message (because of the $(1, w)$ cover-free property)

Source Authentication

- a $(1, w)$-CFF is used to distribute keys for a MAC
- the TA broadcasts a message x, along with v MAC tags for this message: $MAC_{k_1}(x), \ldots, MAC_{k_v}(x)$
- each user U_j verifies the MAC tags using all keys that he possesses
- he accepts the message as being valid only if all the MACs that he can compute are valid
- a coalition F of size at most w cannot create a list of forged MAC tags for a new message x' which will be accepted by any user not in F

Broadcast Encryption to Arbitrary Subsets of Users

- we have seen that a (t, w)-CFF enables broadcast encryption to a subset of at most t users, and a $(1, w)$-CFF enables broadcast encryption to a subset of at least $n - w$ users
- in a general broadcast encryption scheme (BES), a TA wants to broadcast an encrypted message to an **arbitrary** subset P of the n users
- for example, a pay-TV movie, $\mathcal{M}$, might be encrypted with a key K, i.e., $y = e_K(\mathcal{M})$
- the BES is used to encrypt K in such a way that only the members of P can determine K
- the BES is used to encrypt a short key, rather than a long movie, because of the message expansion that is typically required in a BES

The Trivial BES

- suppose the TA gives each user U_i in the network a different key, k_i
- for every $U_i \in P$, the TA can encrypt K with the key k_i, by computing $y_i = e_{k_i}(K)$
- the broadcast $b_P = (y_i : U_i \in P)$ is of length $|P|$, so the **broadcast message expansion** is $|P|$
- this scheme has low storage requirements (one key per user) and high security (no coalition can compute K), but it has a high message expansion
- in general, we want to find good tradeoffs between the parameters of a BES

Designing Efficient BES: A High-Level View

- suppose $P \subseteq \mathcal{U}$, and let w denote the maximum size of a coalition (security parameter)
- many BES involve two steps:
  1. split K into shares $s_1, \ldots, s_v$ using an (r, v) threshold scheme (for some positive integer $r \leq v$)
  2. encrypt every share s_i with a key k_i, in such a way that
     1. every user $U_j \in P$ can compute at least r of the keys $k_1, \ldots, k_v$ (hence they can decrypt r shares of K, and then reconstruct K)
     2. any coalition F, such that $F \cap P = \emptyset$ and $|F| \leq w$, can compute at most $r - 1$ of the keys (hence they can decrypt at most $r - 1$ shares of K, and therefore they cannot obtain any information about K)
- the keys k_i are obtained from key predistribution scheme(s)

Efficient BES

- we construct a collection of v different Fiat-Naor 1-KDPs, each of which is constructed on a certain subset of participants
- a v by n incidence matrix, M, having entries from $\{0, 1\}$, is used to define which users are associated with which schemes
- the v schemes are denoted $\mathcal{F}_1, \ldots, \mathcal{F}_v$
- user U_j is given keys from the KDP $\mathcal{F}_i$ if and only if $M[i, j] = 1$
- for $1 \leq i \leq v$, let

$$\mathit{users}(i) = \{U_j : M[i, j] = 1\}$$

  and for $1 \leq j \leq n$, let

$$\mathit{schemes}(j) = \{i : M[i, j] = 1\}$$

- suppose that $|\mathit{schemes}(j)| = r$ for every j, $1 \leq j \leq n$

Encryption Steps

On input K, the TA performs the following operations:

1. the TA uses the share generation algorithm of an (r, v) threshold scheme to construct v shares, $s_1, \ldots, s_v$
2. for $1 \leq i \leq v$, the TA constructs the group key k_i to be the group key for $P \cap \mathit{users}(i)$ in the scheme $\mathcal{F}_i$
3. for $1 \leq i \leq v$, the TA computes $b_i = e_{k_i}(s_i)$
4. the TA broadcasts the vector $b_P = (b_1, \ldots, b_v)$

Step 2: Constructing the Group Keys

- for $1 \leq i \leq v$, the scheme $\mathcal{F}_i$ is a Fiat-Naor 1-KDP defined on the subset *users*(i)
- to set up $\mathcal{F}_i$, the TA distributes keys as follows:
  1. a key ℓ_i is given to every user in *users*(i)
  2. for every $U_j \in$ *users*(i), a key $\ell_{i,j}$ is given to every user in the set *users*$(i) \backslash \{U_j\}$
- the group key for the subset *users*$(i) \cap P$ is defined to be

$$k_i = \ell_i + \sum_{\{j : U_j \in \textit{users}(i) \backslash P\}} \ell_{i,j}$$

- k_i can be computed (non-interactively) by all members of *users*$(i) \cap P$
- no individual user not in *users*$(i) \cap P$ can compute k_i, but a subset of (at least) two users in *users*$(i) \backslash P$ can compute k_i

Decryption Steps

On input $b_P = (b_1, \ldots, b_v)$, a user $U_j \in P$ performs the following operations:

1. for all $i \in \mathit{schemes}(j)$, U_j constructs the group key k_i for $P \cap \mathit{users}(i)$ in the scheme $\mathcal{F}_i$
2. for all $i \in \mathit{schemes}(j)$, U_j computes $s_i = d_{k_i}(b_i)$
3. U_j uses the share reconstruction algorithm of the (r, v) threshold scheme to compute K from the r shares in the set $\{s_i : i \in \mathit{schemes}(j)\}$

Example of a BES

Suppose $n = 7$, $v = 7$, and the incidence matrix is as follows:

$$M = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{aligned} \mathit{users}(1) &= \{U_1, U_2, U_4\} \\ \mathit{users}(2) &= \{U_2, U_3, U_5\} \\ \mathit{users}(3) &= \{U_3, U_4, U_6\} \\ \mathit{users}(4) &= \{U_4, U_5, U_7\} \\ \mathit{users}(5) &= \{U_1, U_5, U_6\} \\ \mathit{users}(6) &= \{U_2, U_6, U_7\} \\ \mathit{users}(7) &= \{U_1, U_3, U_7\} \end{aligned}$$

Note that every user is associated with $r = 3$ schemes.

Example (cont.)

In the set-up phase, a total of nine keys will be given to each user, as indicated below:

U_1	U_2	U_3	U_4	U_5	U_6	U_7
ℓ_1	ℓ_1	ℓ_2	ℓ_1	ℓ_2	ℓ_3	ℓ_4
$\ell_{1,2}$	$\ell_{1,1}$	$\ell_{2,2}$	$\ell_{1,1}$	$\ell_{2,2}$	$\ell_{3,3}$	$\ell_{4,4}$
$\ell_{1,4}$	$\ell_{1,4}$	$\ell_{2,5}$	$\ell_{1,2}$	$\ell_{2,3}$	$\ell_{3,4}$	$\ell_{4,5}$
ℓ_5	ℓ_2	ℓ_3	ℓ_3	ℓ_4	ℓ_5	ℓ_6
$\ell_{5,5}$	$\ell_{2,3}$	$\ell_{3,4}$	$\ell_{3,3}$	$\ell_{4,4}$	$\ell_{5,1}$	$\ell_{6,2}$
$\ell_{5,6}$	$\ell_{2,5}$	$\ell_{3,6}$	$\ell_{3,6}$	$\ell_{4,7}$	$\ell_{5,5}$	$\ell_{6,6}$
ℓ_7	ℓ_6	ℓ_7	ℓ_4	ℓ_5	ℓ_6	ℓ_7
$\ell_{7,3}$	$\ell_{6,6}$	$\ell_{7,1}$	$\ell_{4,5}$	$\ell_{5,1}$	$\ell_{6,2}$	$\ell_{7,1}$
$\ell_{7,7}$	$\ell_{6,7}$	$\ell_{7,7}$	$\ell_{4,7}$	$\ell_{5,6}$	$\ell_{6,7}$	$\ell_{7,3}$

Example (cont.)

- suppose that the TA wants to broadcast a message to the set $P = \{U_1, U_2, U_3\}$
- the following will be the keys used in the seven Fiat-Naor schemes:

i	$users(i) \cap P$	k_i
1	$\{U_1, U_2\}$	$\ell_1 + \ell_{1,4}$
2	$\{U_2, U_3\}$	$\ell_2 + \ell_{2,5}$
3	$\{U_3\}$	$\ell_3 + \ell_{3,4} + \ell_{3,6}$
4	$\emptyset$	$\ell_4 + \ell_{4,4} + \ell_{4,5} + \ell_{4,7}$
5	$\{U_1\}$	$\ell_5 + \ell_{5,5} + \ell_{5,6}$
6	$\{U_2\}$	$\ell_6 + \ell_{6,6} + \ell_{6,7}$
7	$\{U_1, U_3\}$	$\ell_7 + \ell_{7,7}$

Security (Example)

- we can prove that the example BES is secure against coalitions of size $w = 2$
- let $F = \{U_j, U_{j'}\}$ be an arbitrary coalition of size two
- the incidence matrix M has the property that there is only one set *users*(i) such that $F \subseteq$ *users*(i)
- F can compute only one of the seven group keys
- the threshold $r = 3$ and the coalition can decrypt only one share, so the scheme is secure

Security Properties of Incidence Matrices for BES

- in general, M is required to be a v by n incidence matrix in which there are exactly r "1"s in each column
- let $(\mathcal{U}, \mathcal{A})$ be the set system in which the blocks are formed by the **rows** of M (i.e., the subsets *users*(i))
- note that we previously considered set systems (namely, cover-free families) associated with the **columns** of KDPs
- there are n points and v blocks in $(\mathcal{U}, \mathcal{A})$
- every point (user) is in exactly r blocks (schemes)
- suppose that every pair of points occur in at most λ blocks
- then a coalition of size w can compute **at most** $\lambda\binom{w}{2}$ group keys
- the BES is secure against coalitions of size w if $r > \lambda\binom{w}{2}$

Security Properties of Incidence Matrices (cont.)

- summarizing, we want to construct an incidence matrix such that the associated set system $(\mathcal{U}, \mathcal{A})$ satisfies the following properties:
  1. $|\mathcal{U}| = n$ (there are n points)
  2. $|\mathcal{A}| = v$ (there are v blocks)
  3. every point occurs in exactly r blocks
  4. every pair of points occurs in at most λ blocks
  5. $r > \lambda\binom{w}{2}$
- then M can be used to construct a BES secure against coalitions of size at most w
- clearly, we want v to be small (given n and w)

A Polynomial-based Construction

- let q be prime and let $d \leq q$
- $v = q^2$ and $n = q^d$
- the columns of M are labelled by d-tuples $(a_0, \ldots, a_{d-1}) \in (\mathbb{Z}_q)^d$ (a d-tuple corresponds to a polynomial in $\mathbb{Z}_q[x]$ having degree at most $d-1$)
- the rows of M are labelled by 2-tuples $(x, y) \in (\mathbb{Z}_q)^2$
- the entries of M are defined as follows:

$$M[(x, y), (a_0, \ldots, a_{d-1})] = 1 \Leftrightarrow \sum_{i=0}^{d-1} a_i x^i \equiv y \pmod{q}$$

- it is clear that $r = q$, because every polynomial takes on a unique y-value, given any x-value

Example

Suppose $q = 3$ and $d = 2$; then $n = v = 9$, and M is as follows:

(x, y)	0	1	2	x	$1 + x$	$2 + x$	$2x$	$1 + 2x$	$2 + 2x$
$(0, 0)$	1	0	0	1	0	0	1	0	0
$(0, 1)$	0	1	0	0	1	0	0	1	0
$(0, 2)$	0	0	1	0	0	1	0	0	1
$(1, 0)$	1	0	0	0	0	1	0	1	0
$(1, 1)$	0	1	0	1	0	0	0	0	1
$(1, 2)$	0	0	1	0	1	0	1	0	0
$(2, 0)$	1	0	0	0	1	0	0	0	1
$(2, 1)$	0	1	0	0	0	1	1	0	0
$(2, 2)$	0	0	1	1	0	0	0	1	0

Computing λ

- suppose we have two columns, $(a_0, \ldots, a_{d-1})$ and $(a'_0, \ldots, a'_{d-1})$
- these define two polynomials, $a(x)$ and $a'(x)$, respectively
- we want to determine an upper bound on

$$|\{(x, y) : a(x) = a'(x) = y\}| = |\{x : a(x) = a'(x)\}|$$

- this quantity is at most $d - 1$, because the values at d points determine a **unique** polynomial of degree at most $d - 1$
- hence, $\lambda = d - 1$
- therefore we have $v = q^2$, $n = q^d$, $r = q$ and $\lambda = d - 1$
- the BES is secure against coalitions of size at most w provided that $q > (d-1)\binom{w}{2}$
- if $w = 2$ in the example, then $q = 3 > (d-1)\binom{w}{2} = 1$

Choosing Parameters (Numerical Examples)

- suppose we want to have $w = 3$
- for any prime q, we can take $d < 1 + \frac{q}{3}$
- we get a w-secure BES for $n = q^d$ users, based on $v = q^2$ Fiat-Naor KDPs
- the size of the broadcast is v (elements of $\mathbb{Z}_p$)
- some sample parameters:

q	d	v	n
7	3	49	343
13	5	169	371,293
19	7	361	893,871,739

Choosing Parameters (An Asymptotic Analysis)

- suppose that values for the parameters n and w are given
- define $q \approx w^2 \log_2 n$ and $d \approx \log_2 n$
- $q^d > 2^{\log_2 n} = n$ and $d < q$, so the incidence matrix M exists
- $q = w^2 d > (d-1)\binom{w}{2}$, so the security condition holds
- we obtain a BES secure against coalitions of size w
- the size of the broadcast is $v = q^2 = w^4(\log_2 n)^2$
- for fixed w, v is $O((\log n)^2)$

Storage Requirements

- consider the set system $(X, \mathcal{A})$ corresponding to an incidence matrix M (points correspond to users, and blocks correspond to Fiat-Naor KDPs)
- if $x \in A$, where $A \in \mathcal{A}$, then the corresponding user must store $|A|$ keys from the Fiat-Naor scheme associated with the block A
- in the polynomial scheme, every point is in $r = q$ blocks, and the blocks have size q^{d-1}
- therefore every user stores $q^d = n$ keys in total

An Efficiency Improvement

- in the BES, recall that each user in P can decrypt r shares, and the coalition F can (collectively) decrypt at most $\lambda\binom{w}{2}$ shares, where $\lambda\binom{w}{2} < r$
- the use of an (r, v) threshold scheme ensures that the secret K cannot be computed, given $r - 1$ shares
- if $\lambda\binom{w}{2} < r - 1$, then the security provided by the threshold scheme is stronger than required
- in this situation, it is possible to replace the threshold scheme by a **ramp scheme**
- we can maintain the same security of the BES, while allowing more information to be broadcast

Ramp Schemes

- let t_{low}, t_{high} and n be non-negative integers, $t_{low} < t_{high} \leq n$
- there is a trusted authority, denoted TA, and n users, denoted $U_1, \ldots, U_n$
- the TA has a secret value $K \in \mathcal{K}$
- the TA uses a **share generation algorithm** to split K into n **shares**, denoted $s_1, \ldots, s_n$
- each share $s_i \in \mathcal{S}$, where $\mathcal{S}$ is a specified finite set
- a (t_{low}, t_{high}, n)-**ramp scheme** satisfies the following two properties:
  1. a **reconstruction algorithm** can be used to reconstruct the secret, given any t_{high} of the n shares,
  2. no set of at most t_{low} shares reveals any information as to the value of the secret

Generalized Shamir (Ramp) Scheme

- let $p \geq n+1$ be a prime
- define $t_0 = t_{high} - t_{low}$
- in a threshold scheme, $t_0 = 1$
- let $\mathcal{K} = (\mathbb{Z}_p)^{t_0}$ and let $\mathcal{S} = \mathbb{Z}_p$
- in an initialization phase, $x_1, x_2, \ldots, x_n$ are defined to be n distinct non-zero elements of $\mathbb{Z}_p$
- the TA gives x_i to U_i, for all i, $1 \leq i \leq n$
- the x_i's are public information

Share Generation for the Ramp Scheme

- given a secret $K = (a_0, \ldots, a_{t_0-1}) \in \mathbb{Z}_p$, the share generation algorithm is as follows:
  1. the TA chooses $a_{t_0}, \ldots, a_{t_{high}-1}$ independently and uniformly at random from $\mathbb{Z}_p$
  2. the TA defines
  $$a(x) = \sum_{j=0}^{t_{high}-1} a_j \, x^j$$
  (note that $a(x) \in \mathbb{Z}_p[x]$ is a random polynomial of degree at most $t_{high} - 1$, such that the first t_0 coefficients comprise the secret, K)
  3. for $1 \leq i \leq n$, the TA constructs the share $s_i = a(x_i)$

Reconstruction Algorithm for the Ramp Scheme

- suppose users $U_{i_1}, \ldots, U_{i_{t_{high}}}$ want to determine K
- they know that $s_{i_j} = a(x_{i_j})$, $1 \leq j \leq t_{high}$
- since $a(x)$ is a polynomial of degree at most $t_{high} - 1$, they can determine

$$a(x) = \sum_{j=0}^{t_{high}-1} a_j \, x^j$$

  by Lagrange interpolation
- then $K = (a_0, \ldots, a_{t_0-1})$

Security of the Ramp Scheme

- suppose t_{low} users, say $U_{i_1}, \ldots, U_{i_{t_{low}}}$, pool their shares in an attempt to determine some information about K
- let $(a'_0, \ldots, a'_{t_0-1})$ be a guess for the secret
- then there exists a unique polynomial $a'(x)$ of degree at most $t_{high} - 1$ such that
  1. the first t_0 coefficients of $a'(x)$ are $a'_0, \ldots, a'_{t_0-1}$, and
  2. $s_{i_j} = a'(x_{i_j})$, $1 \leq j \leq t_{low}$
- this is because there are are t_{low} remaining coefficients of $a'(x)$, and the value of $a'(x)$ is known at t_{low} points
- therefore no information about K can be computed

Ramp Schemes and BES

- the BES remains secure if a $\left(\lambda\binom{w}{2}, r, v\right)$ ramp scheme is used to construct shares of the secret
- this allows the size of the secret to be increased: $K \in (\mathbb{Z}_p)^{t_0}$, where

$$t_0 = r - \lambda\binom{w}{2}$$

- shares are still elements of $\mathbb{Z}_p$, however
- therefore the **broadcast ratio** (the length of the broadcast divided by the length of the secret) has been decreased by a factor of t_0, from v to v/t_0

Choosing Parameters (An Asymptotic Analysis)

- as before, define $q \approx w^2 \log_2 n$ and $d \approx \log_2 n$ in the polynomial scheme
- we have

$$\begin{aligned} t_0 &= r - \lambda \binom{w}{2} \\ &= q - (d-1)\binom{w}{2} \\ &\approx w^2 \log_2 n - (\log_2 n)\binom{w}{2} \\ &\approx \frac{w^2 \log_2 n}{2} \end{aligned}$$

- the length of the broadcast is $v = q^2 = w^4 (\log_2 n)^2$
- the broadcast ratio is $O(w^2 \log_2 n)$, which is $O(\log n)$ for fixed w

Multicast Re-keying

- consider the setting of a long-lived dynamic group, say $\mathcal{U}$, with single source broadcast
- an on-line subscription service is one example of this
- the TA wants to broadcast to every user in the group, but members may join or leave the group over time
- communications to the group are encrypted with a single **group key**
- every user has a copy of the group key
- users may also have additional LL-keys, that are used to update the system as the group evolves over time
- the system is initialized in a **key predistribution phase**, during which the TA gives LL-keys and an initial group key to the users in the network

Re-keying Operations

- when a new user joins the group, that user is given a copy of the current group key, as well as appropriate long-term keys (**user join operation**)
- when a user U leaves the group, a **user revocation operation** is necessary to remove the user from the group
- the user revocation operation will establish a new group key for $\mathcal{U} \backslash \{U\}$ (this is sometimes called **re-keying**)
- in addition, updating of LL-keys may be required as part of user revocation operation

Properties of Multicast Re-keying Schemes

Criteria used to evaluate multicast rekeying schemes include the following:

- communication and storage complexity (and trade-offs), for example:
  - size of broadcast, and
  - size (and number) of secret LL-keys required to be stored by users
- security WRT coalitions of revoked users
- anonymity of revoked users
- flexibility/efficiency of user revocation operations, for example:
  - users must be revoked one at a time, or
  - multiple user revocation is possible (up to some specified bound)

Properties of Multicast Re-keying Schemes (cont.)

- flexibility/efficiency of user join operation, for example:
  - any number of new users may be added easily to the system, or
  - entire system must be reinitialized to add new users (one-time system)
- efficiency of updating LL-keys, for example:
  - no updating required (LL-keys are static), or
  - keys can be updated "efficiently", or
  - entire system must be reinitialized after a user revocation (one-time system)

Some Possible Approaches

Some possible approaches to multicast re-keying include the following:

- blacklisting schemes (using $(1, w)$-cover-free families)
- general broadcast encryption schemes
- re-keying schemes based on threshold schemes (Naor-Pinkas)
- tree-based schemes, such as the logical key hierarchy (LKH)

Using a $(1, w)$-CFF to Revoke a Set of w Users

- LL-keys are distributed according to an incidence matrix derived from a $(1, w)$-CFF(v, n)
- let $|F| = w$, and let K denote the new group key for $\mathcal{U} \backslash F$
- for every

$$i \notin \bigcup_{U_j \in F} \mathit{keys}(U_j),$$

  the TA computes $y_i = e_{k_i}(K)$ and broadcasts y_i
- recall that the following properties hold:
  1. no user $U_j \in F$ can compute K (even if the users in F combine all their information)
  2. if $|F| \leq w$, then every user $U_h \notin F$ can decrypt K, because

$$\mathit{keys}(U_h) \not\subseteq \bigcup_{U_j \in F} \mathit{keys}(U_j)$$

Comments

- LL-keys are static in this scheme
- each user stores $O(\log n)$ keys and the broadcast has size $O(\log n)$
- new users can be added to the scheme after it is initialized only if $|\mathcal{U}| < n$
- anonymity is possible, because each user U_j only needs to know the keys in the set *keys*(U_j) (the entire matrix M does not need to be public)
- if it is desired, users can be revoked in stages (up to a total of w revoked users over some period of time) – see next slide
- however, there doesn't appear to be a convenient way to update LL-keys, so the scheme must be re-initialized after w users have been revoked

Revoking Users in Stages

- suppose we want to revoke the users in F_i at stage i, $1 \leq i \leq T$
- we assume that $|F_1| + \cdots + |F_T| \leq w$ and $F_1, \ldots, F_T$ are disjoint
- let K_i denote the group key at stage i, $1 \leq i \leq T$
- the T group keys are broadcast as follows:

Algorithm: *Revoke*$(F_1, \ldots, F_T; K_1, \ldots, K_T)$

$F \leftarrow \emptyset$

for $i \leftarrow 1$ **to** T

do $\begin{cases} F \leftarrow F \cup \{F_i\} \\ \textbf{if } |F| \leq w \\ \quad \textbf{then} \text{ broadcast the group key } K_i \text{ to } \mathcal{U} \backslash F \\ \quad \textbf{else} \text{ quit} \end{cases}$

Revocation Using General BES

- LL-keys are static in this scheme
- each user stores $O(n)$ keys and a broadcast has size $O(\log n)$
- users can be added to the scheme after it is initialized only if $|\mathcal{U}| < n$
- in general, anonymity is not possible
- any number of users can be revoked at any given time
- there is no limit on the total number of users that can be revoked
- however, coalitions of more than w revoked users can compute group keys

Basic Naor-Pinkas Scheme

- denote $n = |\mathcal{U}|$
- the TA uses a $(w+1, n)$ Shamir threshold scheme to construct n shares, say $s_1, \ldots, s_n$, of a new group key, K
- every user U_i is given the share s_i in the intialization phase
- let $|F| = w$ be the set of users to be revoked
- in order to activate the "pre-positioned" group key K, the TA broadcasts the w shares s_i for all $U_i \in F$
- every user **not in** F now has $w+1$ shares, which permits K to be computed
- the users in F are not able to compute F, since they (collectively) hold only w shares

Comments

- LL-keys are static in this scheme
- each user stores $O(1)$ keys and the broadcast has size $O(w)$
- users can be added to the scheme after it is initialized by creation of new shares of K (shares are just evaluations of the secret polynomial at new x-values)
- anonymity is possible, if x-co-ordinates corresponding to the shares are broadcast when the new group key is activated
- it is possible to revoke $w' < w$ users by broadcasting the shares of the w' revoked users, along with $w - w'$ newly created shares that do not correspond to any user
- however, the basic scheme does not allow users to be revoked in stages

A "Re-usable" Version of the Naor-Pinkas Scheme

- suppose that keys and shares are defined in a subgroup G of $\mathbb{Z}_p{}^*$, having prime order q, in which the **Decision Diffie-Hellman** problem is intractable
- let α be a generator of G
- the TA constructs shares of $K \in \mathbb{Z}_q$ using a $(w+1, n)$ Shamir threshold scheme implemented in $\mathbb{Z}_q$
- recall that K can be reconstructed using the formula

$$K = \sum_{j=1}^{w+1} b_j s_{i_j} \bmod q,$$

where the b_j's are coefficients defined as follows:

$$b_j = \prod_{1 \le k \le w+1, k \ne j} \frac{x_{i_k}}{x_{i_k} - x_{i_j}}$$

Re-usable Scheme (cont.)

- then it follows that

$$\alpha^K = \prod_{j=1}^{w+1} \alpha^{b_j s_{i_j}} \bmod p$$

- hence, for any r, it follows that

$$\alpha^{rK} = \prod_{j=1}^{w+1} \alpha^{r b_j s_{i_j}} \bmod p$$

- suppose that the TA broadcasts α^r along with w "exponentiated shares" $\gamma_j = \alpha^{r s_{i_j}}$ $(1 \leq j \leq w)$
- a non-revoked user, say $U_{i_{w+1}}$ can compute his own exponentiated share, as follows:

$$\gamma_{w+1} = \alpha^{r s_{i_{w+1}}} = (\alpha^r)^{s_{i_{w+1}}}$$

Re-usable Scheme (cont.)

- then $U_{i_{w+1}}$ can compute α^{rK}:

$$\alpha^{rK} = \prod_{j=1}^{w+1} \left(\alpha^{rs_{i_j}}\right)^{b_j} \bmod p$$

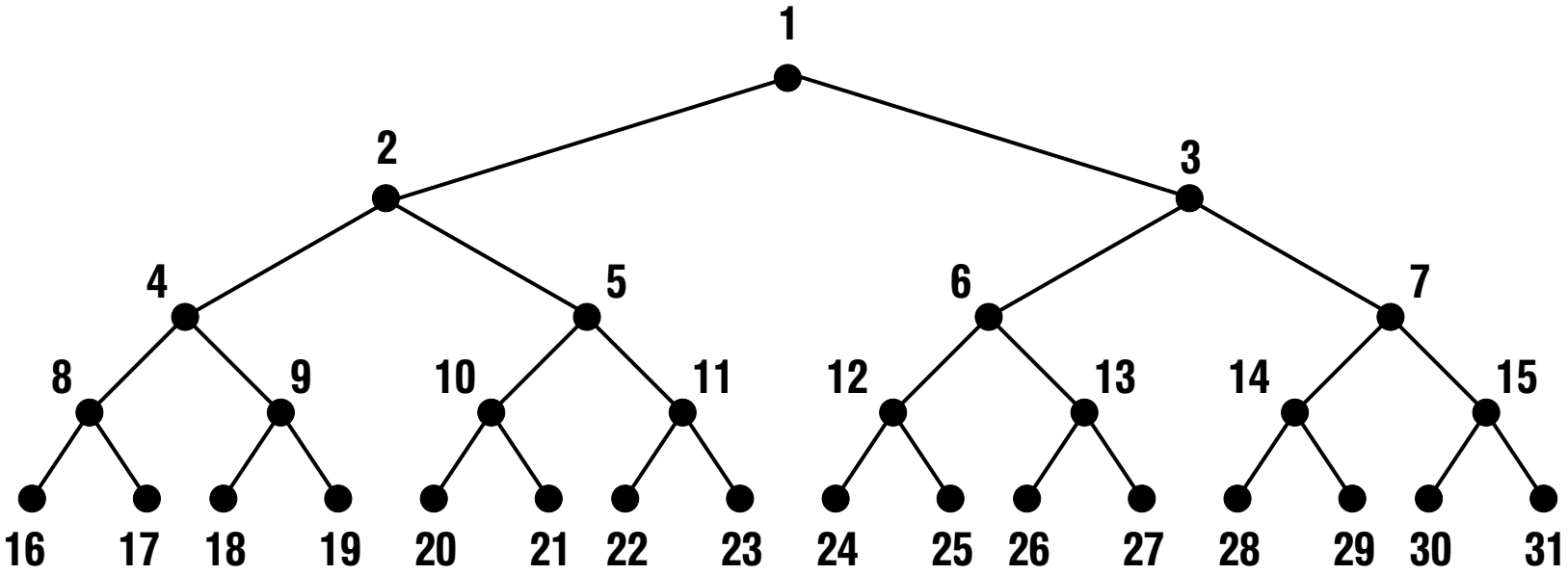
- α^{rK} is the new group key
- by choosing random values of r for each broadcast, a series of new group keys can be set up
- as in previous schemes, coalitions of more than w revoked users can compute group keys
- we would prefer a rekeying scheme that permits an arbitrary number of users to be revoked
- achieving this property apparently requires updating LL-keys when users are revoked

Logical Key Hierarchy (LKH)

- suppose the number of users, n, satisfies $2^{d-1} < n \leq 2^d$
- construct a binary tree, say $\mathcal{T}$, of depth d, having exactly n leaf nodes
- the n leaf nodes of $\mathcal{T}$ correspond to the n users: for every user U, let U also denote the (leaf) node corresponding to the user U
- there is a key associated with **every** node in the tree
- for every node X, let $k(X)$ denote the key for node X
- $k(R)$ is the group key, where R is the root node of $\mathcal{T}$
- every user U is given the $d+1$ keys belonging to the nodes of $\mathcal{T}$ that lie on the unique path from U to R in $\mathcal{T}$
- therefore every user has $O(\log n)$ keys

Example

A binary tree with $d = 4$ and $n = 16$, having nodes labelled $1, 2 \ldots, 2^{d+1} - 1 = 31$:



A Simple Data Structure For Binary Trees

- the leaf nodes are labelled $2^d, 2^d + 1, \ldots, 2^{d+1} - 1$
- the **parent** of node j $(j \neq 1)$ is node $\lfloor \frac{j}{2} \rfloor$
- the **children** of node j $(j \leq 2^d - 1)$ are nodes $2j$ and $2j + 1$
- the **sibling** of node j $(j \neq 1)$ is node

$$\begin{array}{ll} j+1 & \text{if } j \text{ is even} \\ j-1 & \text{if } j \text{ is odd} \end{array}$$

- therefore the binary tree can be stored in the form of an array, $T[1], \ldots, T[2^{d+1} - 1]$, and it is a simple matter to find parents, children and siblings of nodes in this data structure

User Revocation: Removing User U

- let $\mathcal{P}(U)$ denote the set of nodes in the unique path from a leaf node U to the root node R (recall that R has the label 1)
- it is necessary to change the keys corresponding to the d nodes in $\mathcal{P}(U)\backslash\{U\}$
- for each node $X \in \mathcal{P}(U)\backslash\{U\}$, let $k'(X)$ denote the new key for node X
- let $sib(\cdot)$ denote the sibling of a given node, and let $par(\cdot)$ denote the parent of a given node
- the following $2d-1$ items are broadcast by the TA:
  1. $e_{k(sib(U))}(k'(par(U)))$
  2. $e_{k(sib(X))}(k'(par(X)))$ and $e_{k'(X)}(k'(par(X)))$, for all nodes $X \in \mathcal{P}\backslash\{U, R\}$

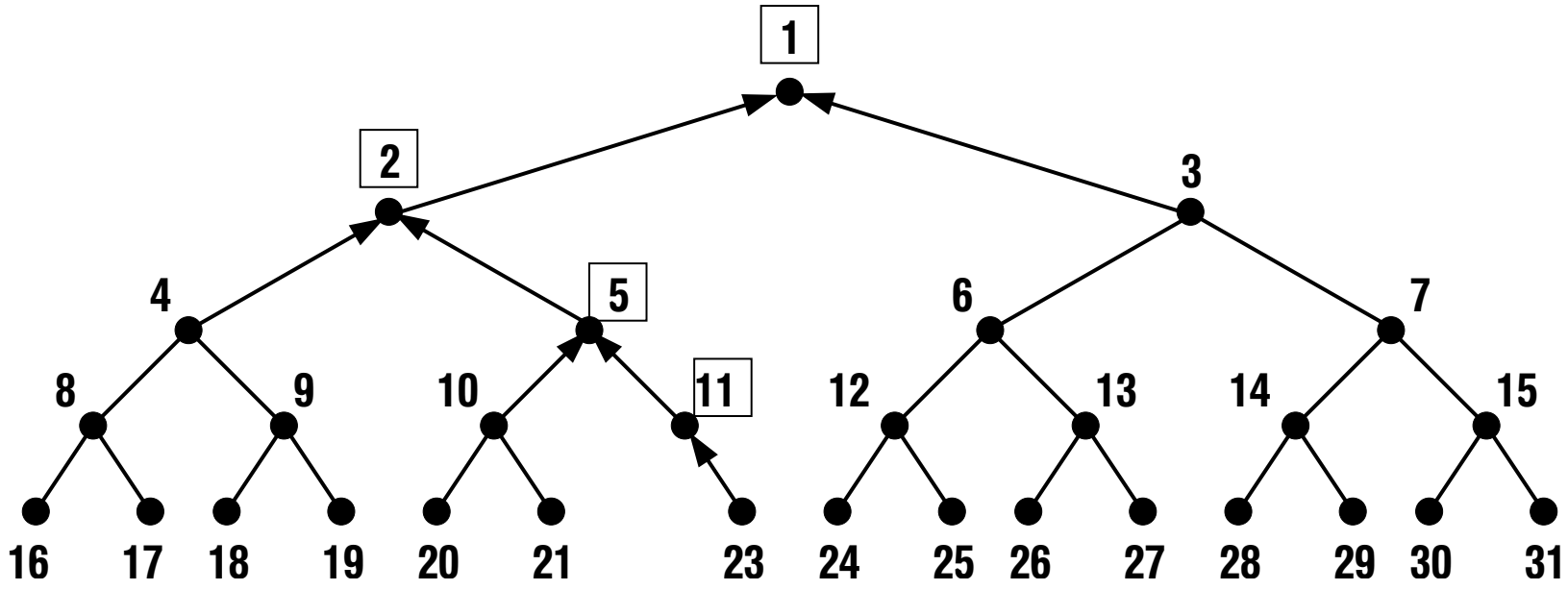
Example

- the broadcast allows any non-revoked user V to update all the keys in the intersection $\mathcal{P}(U) \cap \mathcal{P}(V)$
- for example, suppose the TA wants to revoke user $U = 22$
- the path $\mathcal{P}(U) = \{22, 11, 5, 2, 1\}$
- the TA creates new keys k'_{11}, k'_5, k'_2 and k'_1
- the siblings of the nodes in $\mathcal{P}(U)$ are $\{23, 10, 4, 3\}$
- the broadcast consists of

$$\begin{array}{cccc} e_{k(23)}(k'(11)) & e_{k(10)}(k'(5)) & e_{k(4)}(k'(2)) & e_{k(3)}(k'(1)) \\ & e_{k'(11)}(k'(5)) & e_{k'(5)}(k'(2)) & e_{k'(2)}(k'(1)) \end{array}$$

Example (cont.)

The labels of the nodes receiving new keys are boxed. Encryptions of new keys are indicated by arrows:



Comments

- every user stores $O(\log n)$ keys and the broadcast has size $O(\log n)$
- any number of users can be revoked over a period of time, without affecting the security of the system
- multiple (simultaneous) revocation can be done, but it is somewhat complicated
- new users can be added, whenever the current number of users is less than 2^d, by assigning the new user to an "unoccupied" leaf node of the tree
- if the number of users exceeds 2^d, then a new root node can be created
- this increases the depth d by one, and allows the number of users to be doubled

Copyright Protection

- protection against copyright violation is an important, but very difficult, challenge
- digital content can easily be copied and transmitted over computer networks
- content may be encrypted before it is transmitted
- for example, broadcast encryption protects encrypted content (i.e., unauthorized users cannot decrypt it)
- however, all content must eventually be decrypted in order to be useful
- after content is decrypted, it can potentially be copied
- "hardware-based" solutions provide a limited amount of protection
- other approaches include algorithms for **tracing**

Two Types of Copyright Violation

Here are two potential threats:

illegal content redistribution

Decrypted content can be copied and transmitted to others, for example in an illegal **pirate broadcast**.

illegal key redistribution

Here we are assuming that content is encrypted. The keys used to decrypt the content may be copied and distributed, or they may be used to create a **pirate decoder** which can subsequently be used to illegally decrypt encrypted content.

Fingerprinting

- suppose that every copy of some digital data, D, contains a different **fingerprint**, F
- for example, there might be 1 Mb of binary data, and the fingerprint might consist of 100 "special" bits "hidden" in the data
- the vendor can maintain a database of all the fingerprints, and the owners of the corresponding copies of the data
- then any exact copy can traced back to its owner
- potential problems include the following:
  - if the fingerprint is easily recognized, then it can be modified or destroyed, thus making the data impossible to trace
  - coalitions may be able to recognize fingerprints, even if individual uisers cannot do so

Mathematical Model

- for concreteness, suppose that (a copy of) the data, D, consists of L bits of **content**, say C, and an ℓ-bit **fingerprint**, F, i.e., $D = (C, F)$
- all copies of D have the same content but different fingerprints: $D_1 = (C, F_1), D_2 = (C, F_2), \ldots$
- the fingerprint bits always occur in the same (secret) positions in all copies of the data; e.g., bits $b_{i_1}, \ldots, b_{i_\ell}$ are fingerprint bits
- fingerprinting problems
  are usually studied assuming that a certain **marking assumption** holds:

  > Given some number of copies of the data, say $D_1, D_2, \ldots, D_w$, the only bits that can be identified as fingerprint bits are those bits b such that $D_i[b] \neq D_j[b]$ for some i, j.

Possible Attacks

- the marking assumption implies that the content is irrelevant, and the problem reduces to studying properties of the fingerprints
- given w copies of the data, some bits can be identified as fingerprint bits
- a new "pirate" copy can be constructed, by setting values of these identified bits to be 0 or 1 arbitrarily
- the resulting data $D' = (C, F')$, where F' is a newly created **hybrid fingerprint**
- the fundamental question is whether a hybrid fingerprint can be "traced" if the fingerprints are constructed in a suitable way
- these concepts can be generalized to non-binary data (i.e., data consisting of strings over a specified **alphabet** of size q, say)

Hybrid Fingerprints

- an (ℓ, n, q)-**code** is a subset $\mathcal{C} \subseteq Q^\ell$ such that $|Q| = q$ and $|\mathcal{C}| = n$
- let $\mathcal{C}_0 \subseteq \mathcal{C}$
- define *desc*$(\mathcal{C}_0)$ to consist of all ℓ-tuples $\mathbf{f} = (f_1, \ldots, f_\ell)$ such that, for all $1 \leq i \leq \ell$, there exists a codeword $\mathbf{c} = (c_1, \ldots, c_\ell) \in \mathcal{C}_0$ such that $f_i = c_i$
- for example,

$$\textit{desc}(\{(1,1,2),(2,3,2)\}) = \{(1,1,2),(2,3,2),(1,3,2),(2,1,2)\}$$

- *desc*$(\mathcal{C}_0)$ consists of all the hybrid fingerprints that can be constructed from the fingerprints in $\mathcal{C}_0$
- the codewords in $\mathcal{C}_0$ are called the **parents** of the codewords in *desc*$(\mathcal{C}_0) \backslash \mathcal{C}_0$

Descendant Codes and Suspect Coalitions

- for an integer $w \geq 2$, the **w-descendant code** of $\mathcal{C}$, denoted $desc_w(\mathcal{C})$, consists of the following set of ℓ-tuples:

$$desc_w(\mathcal{C}) = \bigcup_{\mathcal{C}_0 \subseteq \mathcal{C}, |\mathcal{C}_0| \leq w} desc(\mathcal{C}_0)$$

- the w-descendant code consists of all hybrid fingerprints that could be produced by a coalition of size at most w
- suppose that $\mathbf{f} \in desc_w(\mathcal{C})$
- define the set of **suspect coalitions** as follows:

$$susp_w(\mathbf{f}) = \{\mathcal{C}_0 \subseteq \mathcal{C} : |\mathcal{C}_0| \leq w, \mathbf{f} \in desc(\mathcal{C}_0)\}$$

- $\mathcal{C}$ is a **w-identifiable parent property** code (w-IPP code) provided that, for all $\mathbf{f} \in desc_w(\mathcal{C})$, there exists a codeword $\mathbf{c} \in \mathcal{C}$ such that $\mathbf{c} \in \mathcal{C}_0$ for all $\mathcal{C}_0 \in susp_w(\mathbf{f})$

An Example

We present a $(3, 6, 3)$ code, and consider coalitions of size at most 2:

$$\mathbf{c_1} = (0, 1, 1), \quad \mathbf{c_2} = (1, 0, 1), \quad \mathbf{c_3} = (1, 1, 0),$$
$$\mathbf{c_4} = (2, 0, 2), \quad \mathbf{c_5} = (1, 0, 2), \quad \mathbf{c_6} = (2, 1, 0)$$

- consider $\mathbf{f_1} = (1, 1, 1)$
- $\mathit{susp}_2(\mathbf{f_1}) = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 5\}, \{2, 6\}\}$, which violates the 2-IPP property
- consider $\mathbf{f_2} = (2, 1, 2)$
- $\mathit{susp}_2(\mathbf{f_2}) = \{\{1, 4\}, \{3, 4\}, \{4, 6\}, \{5, 6\}\}$, which also violates the 2-IPP property
- the code is not a 2-IPP code

Alternative Definition of IPP Code

- an (ℓ, n, q)-code is a w-IPP code if and only if

$$\bigcap_{\mathcal{C}_0 \in \mathit{susp}_w(\mathbf{f})} \mathcal{C}_0 \neq \emptyset$$

  for all $\mathbf{f} \in \mathit{desc}_w(\mathcal{C})$
- given any hybrid fingerprint $\mathbf{f}$ which was created by a coalition of size at most w in a w-IPP code, any codeword in the above-defined intersection is an **identifiable parent**
- the w-IPP property means that at least one parent is identifiable

An Example

We present a $(3, 7, 5)$ 2-IPP code:

$$\mathbf{c_1} = (0,0,0), \quad \mathbf{c_2} = (0,1,1), \quad \mathbf{c_3} = (0,2,2), \quad \mathbf{c_4} = (1,0,3),$$
$$\mathbf{c_5} = (2,0,4), \quad \mathbf{c_6} = (3,3,0), \quad \mathbf{c_7} = (4,4,0)$$

- suppose that $\mathbf{f} = (f_1, f_2, f_3)$ is a hybrid fingerprint
- if any co-ordinate of $\mathbf{f}$ is non-zero, then at least one parent of $\mathbf{f}$ can be identified:

$$f_1 = 1 \Rightarrow \mathbf{c_4}; \quad f_1 = 2 \Rightarrow \mathbf{c_5}; \quad f_1 = 3 \Rightarrow \mathbf{c_6}; \quad f_1 = 4 \Rightarrow \mathbf{c_7}$$
$$f_2 = 1 \Rightarrow \mathbf{c_2}; \quad f_2 = 2 \Rightarrow \mathbf{c_3}; \quad f_2 = 3 \Rightarrow \mathbf{c_6}; \quad f_2 = 4 \Rightarrow \mathbf{c_7}$$
$$f_3 = 1 \Rightarrow \mathbf{c_2}; \quad f_3 = 2 \Rightarrow \mathbf{c_3}; \quad f_3 = 3 \Rightarrow \mathbf{c_4}; \quad f_3 = 4 \Rightarrow \mathbf{c_5}$$

- finally, if $\mathbf{f} = (0,0,0)$, then $\mathbf{c_1}$ must be a parent

IPP Codes

- in general, it is not an easy task
  - to construct a w-IPP code;
  - to verify whether a given code is a w-IPP code; or
  - to find an efficient algorithm to identify a parent, given an ℓ-tuple in the w-descendant code of a w-IPP code
- it is of interest to design w-IPP codes for which efficient parent-identifying algorithms can be constructed
- we will pursue these questions in the easiest case, $w = 2$

Perfect Hash Families

- an (n, m, w)-**perfect hash family** is a set of functions, say $\mathcal{F}$, such that $|X| = n$, $|Y| = m$, $f : X \to Y$ for each $f \in \mathcal{F}$, and for any $X_1 \subseteq X$ such that $|X_1| = w$, there exists at least one $f \in \mathcal{F}$ such that $f|_{X_1}$ is one-to-one
- when $|\mathcal{F}| = N$, an (n, m, w)-perfect hash family will be denoted by **PHF**$(N; n, m, w)$.
- a PHF$(N; n, m, w)$ can be depicted as an $n \times N$ matrix with entries from Y, having the property that in any w rows there exists **at least one column such that the w entries in the given w rows are distinct**
- perfect hash families have been widely studied in the context of information retrieval algorithms

Separating Hash Families

- an $(n, m, \{w_1, w_2\})$-**separating hash family** is a set of functions, say $\mathcal{F}$, such that $|X| = n$, $|Y| = m$, $f : X \to Y$ for each $f \in \mathcal{F}$, and for any $X_1, X_2 \subseteq X$ such that $|X_1| = w_1$, $|X_2| = w_2$ and $X_1 \cap X_2 = \emptyset$, there exists at least one $f \in \mathcal{F}$ such that

$$\{f(x) : x \in X_1\} \cap \{f(x) : x \in X_2\} = \emptyset$$

- the notation **SHF**$(N; n, m, \{w_1, w_2\})$ will be used to denote an $(n, m, \{w_1, w_2\})$-separating hash family with $|\mathcal{F}| = N$.
- an SHF$(N; n, m, \{w_1, w_2\})$ can be depicted as an $n \times N$ matrix with entries from the set Y, having the property that in any w_1 rows and any w_2 disjoint rows there exists **at least one column such that the entries in the given w_1 rows are distinct from the entries in the given w_2 rows**

An Example

Consider the following 7 by 3 array

0	0	0
0	1	1
0	2	2
1	0	3
2	0	4
3	3	0
4	4	0

- the above array is a PHF$(3; 7, 5, 3)$
- it is also an SHF$(3; 7, 5, \{2, 2\})$
- it is **not** a PHF$(3; 7, 5, 4)$ (consider rows $1, 2, 4$ and 6)

2-IPP Codes

- we will derive an efficient algorithm to determine if a given (ℓ, n, q) code, $\mathcal{C}$, is a 2-IPP code
- suppose the codewords are written in the form of an n by ℓ array, say $A(\mathcal{C})$
- suppose that $A(\mathcal{C})$ is **not** a PHF$(\ell; n, q, 3)$
  - then there exist three rows, r_1, r_2, r_3 of A that violate the PHF property
  - for every column c, let f_c be an element that is repeated (i.e., it occurs in at least two of the three given rows in column c)
  - define $\mathbf{f} = (f_1, \ldots, f_\ell)$
  - clearly $\{r_1, r_2\}, \{r_1, r_3\}, \{r_2, r_3\} \in \mathit{susp}_w(\mathbf{f})$
  - therefore, $\mathcal{C}$ is not a 2-IPP code

2-IPP Codes (cont.)

- suppose that $A(\mathcal{C})$ is **not** an SHF$(\ell; n, q, \{2, 2\})$
  - then there exist two sets of two rows of $A(\mathcal{C})$, $\{r_1, r_2\}$ and $\{r_3, r_4\}$, that violate the SHF property
  - for every column c, let f_c be an element such that it occurs in one of rows r_1 and r_2, and in one of rows r_3 and r_4, in column c
  - define $\mathbf{f} = (f_1, \ldots, f_\ell)$
  - clearly $\{r_1, r_2\}, \{r_3, r_4\} \in \mathit{susp}_w(\mathbf{f})$
  - therefore, $\mathcal{C}$ is not a 2-IPP code
- hence, a **necessary** condition for $\mathcal{C}$ to be a 2-IPP code is that $A(\mathcal{C})$ is simultaneously a PHF$(\ell; n, q, 3)$ and an SHF$(\ell; n, q, \{2, 2\})$

2-IPP Codes (cont.)

- We show that the converse is also true.
- We assume that $\mathcal{C}$ is not a 2-IPP code, and prove that $A(\mathcal{C})$ is not an SHF$(\ell; n, q, \{2, 2\})$ **or** $A(\mathcal{C})$ is not a PHF$(\ell; n, q, 3)$.
- Suppose $\mathbf{f} \in \mathit{desc}_2(\mathcal{C})$ and $\cap_{\mathcal{C}_0 \in \mathit{susp}_2(\mathbf{f})} \mathcal{C}_0 = \emptyset$.
- Let $e = uv \in \mathit{susp}_2(\mathbf{f})$.
- There exists a subset in $\mathit{susp}_2(\mathbf{f})$ not containing u, say $u'v'$.
- If $\{u, v\} \cap \{u', v\} = \emptyset$, then $A(\mathcal{C})$ is not an SHF$(\ell; n, q, \{2, 2\})$.
- Therefore we can assume that $v = v'$.
- There also exists a subset in $\mathit{susp}_2(\mathbf{f})$ not containing v, say $u''v''$.
- If $\{u, v\} \cap \{u', v\} = \emptyset$, then $A(\mathcal{C})$ is not an SHF$(\ell; n, q, \{2, 2\})$.
- Therefore we can assume that $u = u''$.
- If $u' \neq v''$, then the fact that $uv'', u'v \in \mathit{susp}_2(\mathbf{f})$ shows that $A(\mathcal{C})$ is not an SHF$(\ell; n, q, \{2, 2\})$.
- Finally, if $u' = v''$, then the fact that $uv, uv'', vv'' \in \mathit{susp}_2(\mathbf{f})$ shows that $A(\mathcal{C})$ is not a PHF$(\ell; n, q, 3)$.

2-IPP Codes (cont.)

- So we know that $\mathcal{C}$ is a 2-IPP code **if and only if** $A(\mathcal{C})$ is simultaneously a PHF$(\ell; n, q, 3)$ and an SHF$(\ell; n, q, \{2, 2\})$
- as a corollary, an $(\ell, n, 2)$ code cannot be a 2-IPP code if $n \geq 3$
- an (ℓ, n, q) code, $\mathcal{C}$, can be tested to see if it is a 2-IPP code in time $O(n^4\ell)$
- if $\mathcal{C}$ is a 2-IPP code and $\mathbf{f} \in \mathit{desc}_2(\mathcal{C})\backslash\mathcal{C}$, then we can determine $\mathit{susp}_2(\mathbf{f})$ in time $O(n^2\ell)$
- there are two possible cases: either $\mathit{susp}_2(\mathbf{f})$ consists of a single set of two codewords, both of which are identifiable parents; or $\mathit{susp}_2(\mathbf{f})$ consists of a two or more sets of two codewords, all of which contain a fixed codeword (i.e., we have one identifiable parent)

A Construction For 2-IPP Codes with $\ell = 3$

- suppose that $q = r^2 + 2r$
- define

$$
\begin{aligned}
S &= \{1, \ldots, r\} \quad (|S| = r) \\
M &= \{r+1, \ldots, 2r\} \quad (|M| = r) \\
L &= \{2r+1, \ldots, q\} \quad (|L| = r^2) \\
\mathcal{C}_1 &= \{(s_1, s_2, rs_1 + s_2 + r) : s_1, s_2 \in S\} \subseteq S \times S \times L \\
\mathcal{C}_2 &= \{(m, sr + m, s) : m \in M, s \in S\} \subseteq M \times L \times S \\
\mathcal{C}_3 &= \{(rm_1 + m_2 - r^2, m_1, m_2) : m_1, m_2 \in M\} \subseteq L \times M \times M
\end{aligned}
$$

- $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$ is a 2-IPP code with $n = 3r^2$
- this code has an (efficient) $O(1)$ time algorithm to find an identifiable parent

An Example

We present a $(3, 27, 15)$ 2-IPP code:

$$
\begin{array}{lll}
\mathbf{c_1} = (1,1,7), & \mathbf{c_2} = (1,2,8), & \mathbf{c_3} = (1,3,9), \\
\mathbf{c_4} = (2,1,10), & \mathbf{c_5} = (2,2,11), & \mathbf{c_6} = (2,2,12), \\
\mathbf{c_7} = (3,1,13), & \mathbf{c_8} = (3,2,14), & \mathbf{c_9} = (3,3,15), \\
\hline
\mathbf{c_{10}} = (4,7,1), & \mathbf{c_{11}} = (5,8,1), & \mathbf{c_{12}} = (6,9,1), \\
\mathbf{c_{13}} = (4,10,2), & \mathbf{c_{14}} = (5,11,2), & \mathbf{c_{15}} = (6,12,2), \\
\mathbf{c_{16}} = (4,13,3), & \mathbf{c_{17}} = (5,14,3), & \mathbf{c_{18}} = (6,15,3), \\
\hline
\mathbf{c_{19}} = (7,4,4), & \mathbf{c_{20}} = (8,4,5), & \mathbf{c_{21}} = (9,4,6), \\
\mathbf{c_{22}} = (10,5,4), & \mathbf{c_{23}} = (11,5,5), & \mathbf{c_{24}} = (12,5,6), \\
\mathbf{c_{25}} = (13,6,4), & \mathbf{c_{26}} = (14,6,5), & \mathbf{c_{27}} = (15,6,6)
\end{array}
$$

Finding an Identifiable Parent

- if $\mathbf{f} = (f_1, f_2, f_3)$ has a co-ordinate in L, then a parent is easily identified
  - for example, suppose that $r = 3$ (so $q = 15$) and $f_2 = 13$
  - then $3s + m = 13$, where $s, m \in \{1, 2, 3\}$, so $s = 3$, $m = 4$ and hence $(4, 13, 3)$ is an identifiable parent
- if $\mathbf{f}$ has no co-ordinate in L, then it is possible to compute $i \neq j$ such that the two parents of $\mathbf{f}$ are in $\mathcal{C}_i$ and $\mathcal{C}_j$
- the parent that contributed two co-ordinates to $\mathbf{f}$ can then be identified
  - for example, suppose that $\mathbf{f} = (1, 3, 2)$
  - the parents of $\mathbf{f}$ are from $\mathcal{C}_1$ and $\mathcal{C}_2$
  - the parent from $\mathcal{C}_1$ contributes f_1 and f_2
  - hence $(1, 3, 9)$ is an identifiable parent

A Tracing Algorithm

Let (x_1, x_2, x_3) be the input to the tracing algorithm. If $(x_1, x_2, x_3) \in \mathit{desc}_2(\mathcal{C})$, then there are six cases that can occur:

(1) If $x_1 \in L$, compute $m_2 \in M$ such that $m_2 \equiv x_1 \pmod{n}$. Then compute $m_1 = (x_1 + r^2 - m_2)/r$. The codeword (x_1, m_1, m_2) is a parent of (x_1, x_2, x_3).

(2) If $x_2 \in L$, compute $m \in M$ such that $m \equiv x_2 \pmod{n}$. Then compute $s = (x_2 - m)/r$. The codeword (m, x_2, s) is a parent of (x_1, x_2, x_3).

(3) If $x_3 \in L$, compute $s_2 \in S$ such that $s_2 \equiv x_3 \pmod{n}$. Then compute $s_1 = (x_3 - s_2 - r)/r$. The codeword (s_1, s_2, x_3) is a parent of (x_1, x_2, x_3).

A Tracing Algorithm (cont.)

(4) If $x_1, x_2 \in S$, let $s_1 = x_1$ and $s_2 = x_2$. The codeword $(s_1, s_2, rs_1 + s_2 + r) = (x_1, x_2, rx_1 + x_2 + r)$ is a parent of (x_1, x_2, x_3).

(5) If $x_2, x_3 \in M$, let $m_1 = x_2$ and $m_2 = x_3$. The codeword $(rm_1 + m_2 - r^2, m_1, m_2) = (rx_2 + x_3 - r^2, x_2, x_3)$ is a parent of (x_1, x_2, x_3).

(6) If $x_1 \in M$ and $x_3 \in S$, let $m = x_1$ and $s = x_3$. The codeword $(m, sr + m, s) = (x_1, rx_3 + x_1, x_3)$ is a parent of (x_1, x_2, x_3).

(7) If none of the previous six cases arise, then $(x_1, x_2, x_3) \notin \mathit{desc}_2(\mathcal{C})$.

Tracing Illegally Redistributed Keys

- suppose that every user in a network is given a **decoder box** that allows encrypted broadcasts to be decrypted
- i.e., we have a BES in which every user can decrypt the broadcast
- every decoder box contains a different collection of keys
- a coalition of w bad guys might create a **pirate decoder** by combining keys from their decoder boxes
- the keys in each decoder box can be thought of as a codeword in a certain code, and the keys in a pirate decoder can be thought of as a codeword in the w-descendant code
- if the code is traceable (e.g., if it satisfies the w-IPP property), then a pirate decoder can be traced back to at least one member of the coalition that created it

The Broadcast Encryption Scheme

- the TA chooses ℓ sets of keys, denoted $\mathcal{K}_1, \ldots, \mathcal{K}_\ell$, where each $\mathcal{K}_i$ consists of q keys chosen from $\mathbb{Z}_m$, for some fixed m
- for $1 \leq i \leq \ell$, let $\mathcal{K}_i = \{k_{i,j} : 1 \leq j \leq q\}$
- a decoder box contains ℓ keys, one from each set $\mathcal{K}_i$
- the secret key $K \in \mathbb{Z}_p$ (which is used to encrypt the broadcast content, M) is split into ℓ shares using an (ℓ, ℓ) threshold scheme
- the shares are $s_1, \ldots, s_\ell$, where $s_1 + \cdots + s_\ell \equiv K \pmod{m}$
- K is used to encrypt M, and for $1 \leq i \leq \ell$, every $k_{i,j}$ is used to encrypt s_i
- the entire broadcast consists of

$$e_K(M) \quad \text{and} \quad (e_{k_{i,j}}(s_i) : 1 \leq i \leq \ell, 1 \leq j \leq q)$$

Decoder Boxes and Codes

- every user can perform the following operations:
  1. decrypt all ℓ shares of K,
  2. reconstruct K, and
  3. decrypt M
- each decoder box corresponds to a codeword $\mathbf{c} \in Q^\ell$, where $Q = \{1, \ldots, q\}$, in an obvious way:

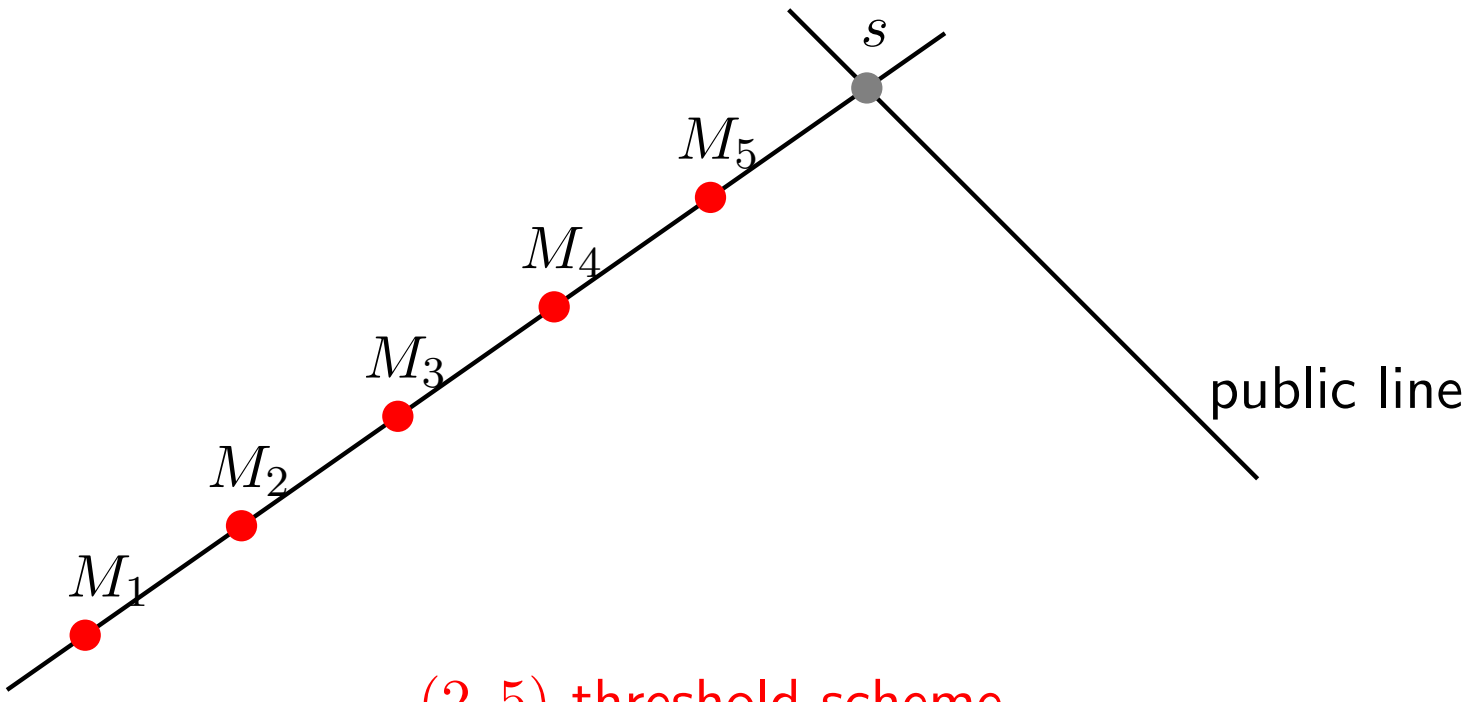
$$\{k_{1,j_1}, k_{2,j_2}, \ldots, k_{\ell,j_\ell}\} \leftrightarrow (j_1, j_2, \ldots, j_\ell)$$

- the keys in a pirate decoder form a codeword in the w-descendant code

Table of Contents

Secret Sharing Scheme

- A bank has 5 managers.
- No single manager is trusted to open the safe.
- Any pair of managers are allowed to open it together.



$(2, 5)$-threshold scheme

(k,n)-Threshold Scheme *(Blakley, Shamir 1979)*

linear scheme: (over GF(p))

$$M = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 4 & \cdots & 2^{k-1} \\ \vdots & & & & \vdots \\ 1 & i & i^2 & \cdots & i^{k-1} \\ \vdots & & & & \vdots \end{pmatrix}$$

secret: s

randomisation:

$\mathbf{r} = (r_1 = s, r_2, \ldots, r_k)$

shares: $M_i \cdot \mathbf{r} = f(i)$

$f(x) =$

$r_1 + r_2 x + r_3 x^2 + \cdots + r_k x^{k-1}$

$$\sum_{j=1}^{k} \alpha_{i_j} M_{i_j} = (1, 0, \ldots, 0) \Rightarrow \sum_{j=1}^{k} \alpha_{i_j} (M_{i_j} \cdot \mathbf{r}) = (1, 0, \ldots, 0) \cdot \mathbf{r} = s$$

M has rank k (Vandermonde)

More General Schemes

Set of participants: $S = \{1, 2, \ldots, n\}$

Definition (monotone access structure)

Collection Σ of subsets of S such that $A' \in \Sigma$ whenever $A' \supseteq A$ and $A \in \Sigma$.

- $A \in \Sigma$ authorised set
- $B \in \Sigma^c := \mathcal{P}(S) \setminus \Sigma$ unauthorised set

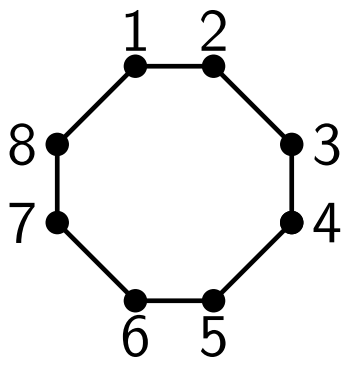
Definition (linear secret sharing scheme realising Σ)

$n' \times d$ matrix M over $GF(p)$ where
$(1, 0, 0, \ldots, 0) \in$ span(rows $I_1, I_2, \ldots, I_j$) iff $\{i_1, i_2, \ldots, i_j\} \in \Sigma$.

Example

$n = 8, n' = 12$

$\Sigma : \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{5,6\},\{7,8\},\{8,1\}\}$



$$M = \left(\begin{array}{ccccc} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 \end{array}\right)$$

Ramp Schemes

Definition (*perfect* secret sharing scheme)

unauthorised sets are unable to determine any information about s

Example ((t, k, n)-ramp scheme)

Take a (k, n)-threshold scheme and define the secret to be $r_1, r_2, r_3, \ldots, r_{k-t}$ (*i.e.* the first $k - t$ coefficients of f).
Then:

- Any k users can recover the secret.
- Any set of at most t users learns no information about the secret.
- If $k > t + 1$, then the ramp scheme is not perfect.

Information Rate

Definition (information rate of a secret sharing scheme)

(size of the secret)/(size of the largest share)

- Every perfect scheme has information rate at most 1.
- An ideal secret sharing scheme has information rate 1.
- Shamir's secret sharing scheme is ideal.
- The previously described (t, k, n)-ramp scheme has (optimal) information rate $k - t$.

(k, n)-Threshold Schemes and Reed-Solomon Codes

$$\mathbf{r} \to f(x) = s + r_2 x + r_3 x^2 + \cdots + r_k x^{k-1}$$
$$\to \text{ shares } (f(1), f(2), \ldots, f(n))$$

The code

$$\mathcal{C} = \{(f(1), f(2), \ldots, f(n)) : f \in \mathrm{GF}(p)[x],\ \deg f < k\}$$

is an $[n, k, n-k+1]$ Reed-Solomon code.

Conclusion: Given the shares of all participants, the secret can be recovered even if $(n-k)/2$ of the shares are corrupted.

Error Correction for General Schemes?

Kaoru Kurosawa: *eprint.iacr.org/2009/263*
General Error Decodable Secret Sharing Scheme and Its Application

e.g. $n = 8$, access structure
$\Sigma: \ \{\{1,2\},\{2,3\},\{3,4\},\{4,5\}\{5,6\},\{6,7\},\{7,8\},\{8,1\}\}$

$(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8)$ share vector

$\downarrow$

$(s_1, s_2, \text{☠}, s_4, \text{☠}, s_6, \text{☠}, s_8)$ corrupt positions of $B \in \Sigma^c$

$\downarrow$

$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8)$ corrupted share vector

Given $(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8)$ can you recover the secret?

General Adversary Structures

- Σ =access structure
- Γ =monotone adversary structure

Definition (monotone adversary structure)

Collection Γ of subsets of S such that $A' \in \Gamma$ whenever $A' \subseteq A$ and $A \in \Gamma$.

Examples:

- $\Gamma = \Sigma^c$ (e.g., as considered by Kurosawa)
- Γ is the collection of subsets of size at most t

Γ-Error Decodable Secret Sharing

Γ-error decodable secret sharing scheme realising an access structure Σ: if shares belonging to members of a set $W \in \Gamma$ are corrupted then the following decoding algorithm succeeds in recovering the correct secret.

Definition (decoding algorithm)

Input: A possibly corrupted share list $\mathbf{t} = (t_1, t_2, \ldots, t_n)$.

1. $\forall$ possible randomisation vectors $\mathbf{r}$ compute the share list $\mathbf{v} = (v_1, v_2, \ldots, v_n) \in \mathrm{GF}(p)^n$.
   If $\{j : v_j \neq t_j\} \in \Gamma$ then r_1 is a candidate secret.
2. If $\exists$ unique candidate secret s, return s.
3. If there are no candidate secrets, or if there is more than one candidate secret, return $\perp$.

A Necessary and Sufficient Condition for Γ-Error Decodability

Definition (condition $Q(\Gamma, \Gamma, \Sigma^c)$)

$\forall\ W_1, W_2 \in \Gamma$, $B \in \Sigma^c$ we have $W_1 \cup W_2 \cup B \neq S$.

Theorem (Fehr-Maurer '02)

A secret sharing scheme is Γ-Error Decodable if and only if condition $Q(\Gamma, \Gamma, \Sigma^c)$ is satisfied.

Γ-Error Decodability (cont.)

Proof:

Γ-Error Decodability (cont.)

Proof: $(\Rightarrow)$:

$$\begin{array}{cccc} & W_1 & W_2 & B \\ \mathbf{v}^2 & \boxed{X'} & \boxed{Y'} & \boxed{Z} \end{array} \rightarrow s_2 \neq s_1$$

$$\mathbf{v}^1 \quad \boxed{X} \boxed{Y} \boxed{Z} \rightarrow s_1$$

Γ-Error Decodability (cont.)

Proof: $(\Rightarrow)$:

$$
\begin{array}{cccc}
 & W_1 & W_2 & B \\
\mathbf{v}^2 & X' & Y' & Z \quad \rightarrow s_2 \neq s_1 \\
 & & \downarrow & \\
\mathbf{t} & X & Y' & Z \quad \rightarrow \perp \\
 & & \uparrow & \\
\mathbf{v}^1 & X & Y & Z \quad \rightarrow s_1
\end{array}
$$

Γ-Error Decodability (cont.)

Proof: $(\Leftarrow)$:

$$W_1 \, W_2 \;\; B$$

$$\mathbf{t} \; \boxed{X \mid Y' \mid Z} \; \rightarrow \perp$$

Γ-Error Decodability (cont.)

Proof: $(\Leftarrow)$:

$$
\begin{array}{cccc}
 & W_1 & W_2 & B \\
\mathbf{v}^2 & X' & Y' & Z \quad \rightarrow s_2 \neq s_1 \\
 & & \uparrow & \\
\mathbf{t} & X & Y' & Z \quad \rightarrow \perp \\
 & & \downarrow & \\
\mathbf{v}^1 & X & Y & Z \quad \rightarrow s_1
\end{array}
$$

Efficiency of Error Decoding

- Generating the shares for any linear secret-sharing scheme is efficient.
- There are efficient algorithms for decoding Reed-Solomon codes.
- For adversary structures other than the threshold case it is not generally known whether there exists an error decodable secret sharing scheme with efficient decoding.

Kurosawa's Polynomial Time Error Decodable Scheme (Generalisation)

Takes any linear Σ^c-error decodable secret sharing scheme and constructs a Σ^c-error decodable secret sharing scheme with polynomial time decoding*, but having larger shares.

Kurosawa's Polynomial Time Error Decodable Scheme (Generalisation)

Takes any linear Σ^c-error decodable secret sharing scheme and constructs a Σ^c-error decodable secret sharing scheme with polynomial time decoding*, but having larger shares.

* polynomial in the total size of the shares. If the total size of the shares is polynomial in the number of participants, (*e.g.* for an ideal scheme) then Kurosawa's scheme can be decoded in time polynomial in the number of participants.

Kurosawa's Polynomial Time Scheme

$$\begin{array}{cccccc} & & & s & & \\ & & & \downarrow & & \\ v^1 & v^2 & v^3 & v^4 & \cdots & v^n \end{array}$$

level 1

M is used to generate share vector $\mathbf{v}$ corresponding to secret s

Kurosawa's Polynomial Time Scheme

$$\begin{array}{cccccc} & & s & & & \\ & & \downarrow & & & \\ \mathbf{w}^1 & v^2 & v^3 & v^4 & \cdots & v^n \\ \boxed{u_1^1} & \boxed{u_2^1} & \boxed{u_3^1} & \boxed{u_4^1} & \cdots & \boxed{u_n^1} \end{array}$$

level 1 M is used to generate share vector $\mathbf{v}$ corresponding to secret s

level 2 For $i = 1, 2, \ldots, n$ share v^i is converted to new secret vector $\mathbf{w}^i$ and M is used to generate corresponding share vector $\mathbf{u}^i$. Note: $\mathbf{w}^i$ includes the randomness used to generate $\mathbf{u}^i$.

Kurosawa's Polynomial Time Scheme

$$
\begin{array}{cccccc}
 & & s & & & \\
 & & \downarrow & & & \\
\mathbf{w}^1 & \mathbf{w}^2 & v^3 & v^4 & \cdots & v^n \\
\hline
u_1^1 & u_2^1 & u_3^1 & u_4^1 & \cdots & u_n^1 \\
\hline
u_1^2 & u_2^2 & u_3^2 & u_4^2 & \cdots & u_n^2 \\
\hline
\end{array}
$$

level 1 M is used to generate share vector $\mathbf{v}$ corresponding to secret s

level 2 For $i = 1, 2, \ldots, n$ share v^i is converted to new secret vector $\mathbf{w}^i$ and M is used to generate corresponding share vector $\mathbf{u}^i$. Note: $\mathbf{w}^i$ includes the randomness used to generate $\mathbf{u}^i$.

Kurosawa's Polynomial Time Scheme

$$
\begin{array}{cccccc}
 & & s & & & \\
 & & \downarrow & & & \\
\mathbf{w}^1 & \mathbf{w}^2 & \mathbf{w}^3 & \mathbf{w}^4 & \cdots & \mathbf{w}^n \\
\hline
u_1^1 & u_2^1 & u_3^1 & u_4^1 & \cdots & u_n^1 \\
\hline
u_1^2 & u_2^2 & u_3^2 & u_4^2 & \cdots & u_n^2 \\
\hline
u_1^3 & u_2^3 & u_3^3 & u_4^3 & \cdots & u_n^3 \\
\hline
 & & \vdots & & & \\
\hline
u_1^n & u_2^n & u_3^n & u_4^n & \cdots & u_n^n \\
\hline
\end{array}
$$

level 1

level 2

M is used to generate share vector $\mathbf{v}$ corresponding to secret s

For $i = 1, 2, \ldots, n$ share v^i is converted to new secret vector $\mathbf{w}^i$ and M is used to generate corresponding share vector $\mathbf{u}^i$. Note: $\mathbf{w}^i$ includes the randomness used to generate $\mathbf{u}^i$.

Participant j receives share $\bigcup_{i=1}^{n} u_j^i \cup \mathbf{w}^j$, i.e., the jth column of data.

Kurosawa's Polynomial Time Scheme -Efficient Decoding

1. $\forall\ i$, generate share vector corresponding to secret vector $\mathbf{w}^i$, compare with other participants' level 2 shares.
2. If the set of positions where they differ is not in Γ, conclude that $\mathbf{w}^i$ is corrupted.
   Note: This can be done efficiently if $\Gamma = \Sigma^c$ because the scheme is linear.
3. Use uncorrupted level 1 shares to recover s.

received shares

$$\begin{array}{cccccc} \mathbf{w}^1 & \mathbf{w}^2 & \mathbf{w}^3 & \mathbf{w}^4 & \cdots & \mathbf{w}^n \\ \hline \boxed{t_1^1} & \boxed{t_2^1} & \boxed{t_3^1} & \boxed{t_4^1} & \cdots & \boxed{t_n^1} \\ \hline \boxed{t_1^2} & \boxed{t_2^2} & \boxed{t_3^2} & \boxed{t_4^2} & \cdots & \boxed{t_n^2} \\ & & & \vdots & & \end{array} \quad =? \quad \begin{array}{cccccc} & & \mathbf{w}^2 & & & \\ & & \downarrow & & & \\ \boxed{u_1^2} & \boxed{u_2^2} & \boxed{u_3^2} & \boxed{u_4^2} & \cdots & \boxed{u_n^2} \end{array}$$

Reducing the Storage Requirements of Kurosawa's Scheme

How to reduce the size of the level 2 shares:

- The level 2 schemes need not be perfect; they are only used to authenticate the level 1 shares.
- It suffices for the level 2 shares to be assigned using any (possibly non-perfect) secret-sharing scheme with the following properties:
  1. Sets of participants in Σ^c learn no information about the secret.
  2. For any two adversary sets $W_1, W_2 \in \Gamma$, the participants in $S \setminus (W_1 \cup W_2)$ should be able to recover the secret (this property is required to ensure that a level 2 share list, corrupted by an adversary set in Γ, determines a unique level 1 secret).
- Often, we can replace M by an appropriate ramp scheme.

Reducing the Storage Requirements of Kurosawa's Scheme (cont.)

How to reduce the number of level 2 schemes required:

- $A \subseteq S :=$ participants whose level 1 shares are shared using level 2 schemes.
- Decoding succeeds if we can find an authorised set whose shares are confirmed to be uncorrupted:

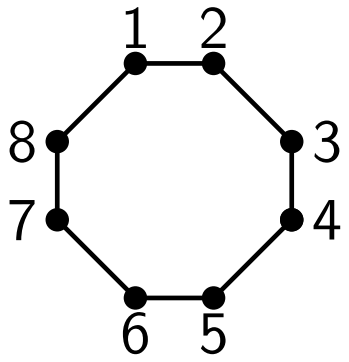
$$\forall W \subseteq A \text{ with } W \in \Gamma \text{ we have } A \setminus W \in \Sigma.$$

Corollary: The number of level 2 schemes required is upper bounded by

$$1 + \max_{W \in \Gamma} |W| + \max_{B \in \Sigma^c} |B|.$$

Example

$n = 8$, $\Sigma : \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{5,6\},\{7,8\},\{8,1\}\}$



Γ: single participants

- It suffices to provide level 2 sharings for $\{1,2,3,4\}$ (given one adversary in $\{1,2,3,4\}$, there is still an uncorrupted authorised set in $\{1,2,3,4\}$).
  (This cuts the number of level 2 schemes needed by half.)
- We can use a $(4,6,8)$-ramp scheme ($|S \backslash (W_1 \cup W_2)| = 6$, and the maximum size of an unauthorised subset is 4).
  (This requires at most half the storage of any perfect scheme.)

One-Round (n, t)-Perfectly Secure Message Transmission *(Dolev, Dwark, Waarts, Yung 1993)*

Alice transmits a message s to Bob by sending information over n channels so that:

- Bob recovers s even if Eve corrupts $\leq t$ of the channels;
- Eve learns no information about s from the information Alice sent on the channels she corrupts.

- A (n, t)-PSMT scheme exists iff $n \geq 3t + 1$. (Dolev *et al.*)
- Desmedt, Wang and Burmester (2005):
  If Eve corrupts channels corresponding to a set in Γ then one-round PSMT is possible iff condition $Q(\Gamma, \Gamma, \Gamma)$ holds.
- When Γ is a threshold structure, the Dolev *et al* result is recovered.

One-Round (Γ, Σ^c)-PSMT

We consider a more general setting:

- Bob correctly recovers s if the information sent on a set $W \in \Gamma$ of channels is changed.
- Eve learns nothing about s if she eavesdrops on a set $D \in \Sigma^c$ of channels.

Theorem

A one-round (Γ, Σ^c)-PSMT scheme exists iff condition $Q(\Gamma, \Gamma, \Sigma^c)$ holds.

Proof: $(\Leftarrow)$: Use a Γ-error decodable secret sharing scheme realising Σ, send a share down each channel!
$(\Rightarrow)$: Use the proof technique from the error-decodability theorem.
Corollary: A one-round (Γ, Σ^c)-PSMT scheme exists iff there exists a Γ-error decodable secret sharing scheme realising Σ.

So are they really just the same thing?

So are they really just the same thing?

Not quite...

So are they really just the same thing?

Not quite...

Theorem

A one-round (Γ, Σ^c)-PSMT scheme is equivalent to a (not necessarily perfect) secret-sharing scheme where

- *the authorised sets are those of the form $S \setminus (W_1 \cup W_2)$ with $W_1, W_2 \in \Gamma$,*
- *the unauthorised sets belong to Σ^c.*

So are they really just the same thing?

Not quite...

Theorem

A one-round (Γ, Σ^c)-PSMT scheme is equivalent to a (not necessarily perfect) secret-sharing scheme where

- *the authorised sets are those of the form $S \setminus (W_1 \cup W_2)$ with $W_1, W_2 \in \Gamma$,*
- *the unauthorised sets belong to Σ^c.*

Corollary: A one-round (n, t)-PSMT scheme is equivalent to a $(t, n - 2t, n)$-ramp scheme.

Efficiency of One-Round PSMT: Number of Channels

S -set of channels, Γ -active adversary, Σ^c -passive adversary

The minimum number of channels needed for one-round (Γ, Σ^c)-PSMT is $|T|$, where $T \subseteq S$ is the smallest subset for which $Q(\Gamma_T, \Gamma_T, \Sigma^c_T)$ holds.
Note: Γ_T denotes the restriction of Γ to T, and Σ^c_T denotes the restriction of Σ^c to T.

Corollary:

$$|T| \leq 1 + 2\max_{W \in \Gamma}|W| + \max_{B \in \Sigma^c}|B|.$$

(In the threshold case this reproves the result that one-round (n, t)-PSMT is possible iff $n \geq 3t + 1$.)

Efficiency of One-Round PSMT: Transmitted Info

Definition (overhead)

(total information sent over all channels)/(size of message s)

- Desmedt *et al.* describe a construction for a one-round (Γ, Γ)-PSMT for any Γ satisfying $Q(\Gamma, \Gamma, \Gamma)$ that's equivalent to a known secret sharing scheme construction.
- Kurosawa points out that in the threshold case this has a worse overhead than if an ideal threshold scheme is used.
- You can do better still if you use a ramp scheme!

Corollary (Fitzi *et al*): The optimal overhead of a one-round (n, t)-PSMT scheme is $n/(n-3t)$.
Proof: Use the equivalence with ramp schemes and the fact that the optimal information rate of a (t, k, n)-ramp scheme is $k - t$ (Jackson & Martin).

Open Problems

- Do there exist constructions of one-round (Γ, Σ^c)-PSMT schemes with polynomial time message recovery for general Γ, Σ with lower communication overheads?
- Is it possible to determine in general which classes of Γ and Σ can be realised by schemes with efficient decoding/message recovery?
- Is it possible to find efficient decoding/message recovery techniques for specific classes of Γ and Σ?

POR schemes

- Alice asks a *server* to store a (possibly large) file (or message) m (e.g., using cloud storage).
- The message m is divided into message blocks that we view as elements of a finite field.
- Typically, the message m will be encoded as M, using a public error-correcting code such as a Reed-Solomon code.
- The code provides redundancy, enabling erasures or corrupted message blocks in M to be corrected.
- Main problem: How can Alice be convinced that the *server* is storing the encoded message M correctly?
- Typical solution: A challenge-response protocol is periodically invoked by Alice.

Bounded-use schemes

- We do not assume that Alice is storing m or M.
- Alice must precompute and store a fixed number of challenge-response pairs, before transmitting M to the *server*.
- Alice gains confidence in the *server* if it is able to respond to all (or most of) her challenges.
- A *server* who can respond correctly to a large proportion of challenges should "know" (or be able to compute) the contents of the unencoded message m (i.e., all the message blocks).
- This idea is formalised in the notion of an extractor, in which case we have a proof-of-retrievability (or POR) scheme.

Extractors

- The *Extractor* takes as input a description of the *server*'s proving algorithm, denoted $\mathcal{P}$, and then outputs an unencoded message $\widehat{m}$.
- Extraction succeeds if $\widehat{m} = m$.
- The success probability of $\mathcal{P}$, denoted succ($\mathcal{P}$), is the probability that $\mathcal{P}$ gives a correct response for a randomly chosen challenge.
- Definition: the POR scheme is (δ, ϵ)-secure if the *Extractor* succeeds with probability at least δ whenever succ($\mathcal{P}$) $\geq \epsilon$.

Some previous related work

- Blum *et al.* (1994) introduced memory checking.
- Lillibridge *et al.* (2005) studied internet backup schemes.
- Naor and Rothblum (2005) studied online memory checkers and authenticators and they gave a lower bound on storage requirements and communication complexity.
- Juels and Kaliski (2007) introduced proof of retrievability schemes.
- Atieniese *et al.* (2007) introduced proof of data possession schemes.
- Shacham and Waters (2008) gave examples of unbounded-use schemes along with formal security proofs.
- Bowers, Juels, and Oprea (2009) used inner and outer codes to construct POR schemes.
- Dodis, Vadhan and Wichs (2009) gave the first examples of unconditionally secure POR schemes.

Three phases in a POR scheme

1. initialisation

Alice		*server*
$M = e(m)$	$\xrightarrow{M}$	*server* constructs $\mathcal{P}$

Three phases in a POR scheme

1. initialisation

Alice		server
$M = e(m)$	$\xrightarrow{M}$	*server* constructs $\mathcal{P}$

2. audit

Alice		server
	$\xrightarrow{c_i}$	
		$r_i = \mathcal{P}(c_i)$
	$\xleftarrow{r_i}$	

Here $i = 1, 2, \ldots$.

Three phases in a POR scheme

1. initialisation

Alice — server

$M = e(m) \xrightarrow{M}$ *server* constructs $\mathcal{P}$

2. audit

Alice — server

$\xrightarrow{c_i}$

$r_i = \mathcal{P}(c_i)$

$\xleftarrow{r_i}$

3. extraction

Alice — *server* — *Extractor*

$\xrightarrow{\mathcal{P}}$

$\xleftarrow{\widehat{m}}$

Our problem setting

- We study POR schemes in the setting of unconditional security, where the adversary is assumed to have unlimited computational capabilities.
- We only consider POR schemes where $\delta = 1$, that is, where extraction is guaranteed to be successful.
- The constructions that we utilise for extractors only require black-box access to the proving algorithm.
- In this setting, it turns out that extraction can be interpreted naturally as nearest-neighbour decoding in a certain code (which we term a response code).
- Error-correcting codes have been used in many constructions of POR schemes; we propose that error-correcting codes constitute the natural foundation to construct as well as analyse arbitrary POR schemes.

Why unconditional security?

- Simplicity and mathematical elegance: The schemes are mathematically elegant as well as easier to understand and analyse because we are not making use of any additional cryptographic primitives.
- Exact analyses: We can give very simple exact (i.e., non-asymptotic) analyses of various schemes.
- Links with error-correcting codes: The essential role of error-correcting codes in the design and analysis of POR schemes becomes clear: codes are not just a method of constructing POR schemes; rather, every POR scheme gives rise to a code in a natural way.
- Adversarial strength: It is interesting and informative to consider security against the strongest possible adversary and to prove security results that do not depend on unproven assumptions.

Basic Scheme

Initialisation

Given a message $m \in (\mathbb{F}_q)^k$, encode M as $e(m) = M \in (\mathbb{F}_q)^n$, where q is a prime power and $n \geq k$. The set of encoded messages is the encoded message space. We write $M = (m_1, \ldots, m_n)$.

Alice gives M to the *server*. Alice also generates a random challenge $c \in \{1, \ldots, n\}$ and she stores c and m_c.

Challenge-response

Alice gives the challenge c to the *server*. The *server* responds with $r = m_c$. Alice checks that $r = m_c$.

The extractor

1. Compute responses to all possible challenges: On input $\mathcal{P}$, compute the response vector $M' = (m'_1, \ldots, m'_n)$, where $m'_c = \mathcal{P}(c)$ for all $c \in \{1, \ldots, n\}$ (i.e., m'_c is the response from $\mathcal{P}$ when it is given the challenge c).
2. Nearest-neighbour decoding: Find an encoded message $\widehat{M}$ so that $\mathrm{dist}(M', \widehat{M})$ is minimised, where $\mathrm{dist}(\cdot, \cdot)$ denotes the hamming distance between two vectors.
3. Output $\widehat{m} = e^{-1}(\widehat{M})$.

Theorem

Theorem

Suppose that $\mathcal{P}$ is a proving algorithm for the Basic Scheme for which

$$\mathsf{succ}(\mathcal{P}) > 1 - \frac{d}{2n},$$

where the hamming distance of the encoded message space is d. Then the Extractor will always output $\widehat{m} = m$.

Example

- Suppose that Alice wants to use the Basic Scheme with $q = 2^{10}$ and $n = 1000$ such that the minimum distance of the encoded message space is 400.
- This will guarantee that extraction will be possible whenever $\mathsf{succ}(\mathcal{P}) > 0.8$.
- If Alice uses a Reed-Solomon code to encode messages, then $d = n - k + 1$, where k is the dimension of the code.
- Therefore, $k = 601$, so the message expansion is

$$\frac{1000}{601} \approx 1.67.$$

- The size of a challenge is $\log_2 n = 10$ bits and the size of a response is $\log_2 q = 10$ bits.

Generalisation

We can consider arbitrary challenge-response protocols, where a challenge will be chosen from a specified challenge space Γ, and the response will be an element of a response space Δ. The response code consists of all $|\Gamma|$-tuples of elements from Δ that are obtained as correct responses for some encoded message M. We can prove a straightforward generalisation of the previous theorem.

Theorem

Suppose that $\mathcal{P}$ is a proving algorithm for a General POR Scheme for which

$$\mathsf{succ}(\mathcal{P}) > 1 - \frac{d^*}{2|\Gamma|},$$

where the hamming distance of the response code is d^*. Then the Extractor based on nearest neighbour decoding will always output $\widehat{m} = m$.

Multiblock Challenge Scheme

- Here, a challenge specifies ℓ indices "all at once", say $i_1 < \cdots < i_\ell$.
- $|\Gamma| = \binom{n}{\ell}$.
- The response is the ℓ-tuple $(m_{i_1}, \ldots, m_{i_\ell})$.
- If the hamming distance of the encoded message space is d, then the hamming distance of the response code is

$$d^* = \binom{n}{\ell} - \binom{n-d}{\ell}.$$

- Therefore, extraction succeeds if

$$\mathsf{succ}(\mathcal{P}) > \frac{1}{2} + \frac{\binom{n-d}{\ell}}{2\binom{n}{\ell}}.$$

Linear Combination Scheme

- A challenge V is a vector in $(\mathbb{F}_q)^n$ having hamming weight equal to ℓ.
- The response is

$$V \cdot M = \sum_{i=1}^{n} v_i m_i \bmod q.$$

- $|\Gamma| = \binom{n}{\ell}(q-1)^\ell$ and $|\Delta| = q$.
- If the hamming distance of the encoded message space is d, then a very accurate estimate for the hamming distance of the response code is

$$d^* \approx \frac{(q-1)^{\ell+1}}{q}\left(\binom{n}{\ell} - \binom{n-d}{\ell}\right).$$

- Therefore, extraction succeeds if

$$\mathsf{succ}(\mathcal{P}) > \frac{1}{2} + \frac{1}{2}\left(\frac{1}{q} + \frac{(q-1)\binom{n-d}{\ell}}{q\binom{n}{\ell}}\right).$$

Comparison

- The Linear Combination Scheme has much smaller responses than the Multiblock Challenge Scheme ($\mathbb{F}_q$ as opposed to $(\mathbb{F}_q)^\ell$).
- However, the Linear Combination Scheme has a larger challenge space than the Multiblock Challenge Scheme ($\binom{n}{\ell}(q-1)^\ell$ as opposed to $\binom{n}{\ell}$).
- The relative distance of the response codes of the two schemes are very similar, so the security guarantees of the two schemes are also very similar.

Example

- Suppose that Alice wants to use the Linear Combination Scheme with $q \geq 2^{10}$ and $n = 1000$.
- Her goal is that extraction will be possible whenever $\mathsf{succ}(\mathcal{P}) > 0.8$.
- Here, $d = 50$ and $\ell = 10$ will work.
- If Alice uses a Reed-Solomon code to encrypt messages, then $k = 951$, so the message expansion is

$$\frac{1000}{951} \approx 1.05.$$

- The size of a challenge is 178 bits and the size of a response is $\log_2 q = 10$ bits.

Estimating the success probability of a prover

- We have proven that extraction is possible provided that $\mathsf{succ}(\mathcal{P})$ is sufficiently close to 1.
- In general, the only way to determine the exact value of $\mathsf{succ}(\mathcal{P})$ is to query $\mathcal{P}$ with all the possible challenges (as is done during extraction).
- In practice, we would like to be able to estimate $\mathsf{succ}(\mathcal{P})$ based on a relatively small number of challenges.
- This can be done using classical statistical techniques such as hypothesis testing and confidence intervals.

Hypothesis testing for the Basic Scheme

- We know that extraction will be successful in the Basic Scheme if

$$\mathsf{succ}(\mathcal{P}) \geq \frac{n - \lfloor \frac{d}{2} \rfloor + 1}{n}.$$

- Denote $\omega = n - \lfloor \frac{d}{2} \rfloor + 1$.
- We wish to distinguish the null hypothesis

$$H_0 : \mathsf{succ}(\mathcal{P}) \leq \frac{\omega - 1}{n};$$

from the alternative hypothesis

$$H_1 : \mathsf{succ}(\mathcal{P}) \geq \frac{\omega}{n}.$$

- If we reject the null hypothesis H_0, then we believe that extraction is possible.

Hypothesis testing for the Basic Scheme (cont.)

- Suppose there are g correct responses in t trials.
- For simplicity, assume the challenges are chosen uniformly at random with replacement.
- The condition for rejecting the null hypothesis at a 5% significance level is

$$\sum_{i=g}^{t} \binom{t}{i} \left(\frac{\omega - 1}{n}\right)^{i} \left(\frac{n - \omega + 1}{n}\right)^{t-i} < 0.05.$$

- If this condition holds, then we are quite confident that successful extraction is possible.

Example

- Suppose that Alice using the Basic Scheme with $n = 1000$ and the minimum distance of the encoded message space is 400.
- Then extraction is possible whenever succ$(\mathcal{P}) > 0.8$.
- Suppose the *server* responds to 100 challenges that have been chosen uniformly with replacement, and that 87 of the responses were correct.
- We find that

$$\sum_{i=87}^{100} \binom{100}{i} 0.8^i \, 0.2^{100-i} \approx 0.047 < 0.05.$$

- There is sufficient evidence to reject the null hypothesis at the 5% significance level, and so we conclude that the file can be reconstructed by an extractor.

A new lower bound on storage and communication

- Suppose that $\mathbf{M}$ is a random variable corresponding to a randomly chosen unencoded message m.
- Let $\mathbf{V}$ be a random variable denoting any information stored by Alice
- Let $\mathbf{R}$ be a random variable corresponding to the information provided by a black-box *Extractor*.
- Suppose that the message can be reconstructed by the *Extractor* with probability 1
- Then

$$H(\mathbf{M}|\mathbf{V},\mathbf{R}) = 0,$$

from which it follows that

$$H(\mathbf{M}) \leq H(\mathbf{V}) + H(\mathbf{R}).$$

Lower bound (cont.)

- Naor and Rothblum proved a lower bound for a weaker form of POR-type protocol, termed an an authenticator.
- The Naor-Rothblum bound also applies to POR schemes.
- Phrased in terms of entropy, their bound states that

$$H(\mathbf{M}) \leq H(\mathbf{V}) \times H(\mathbf{R}),$$

which is a weaker bound than the one we proved above.