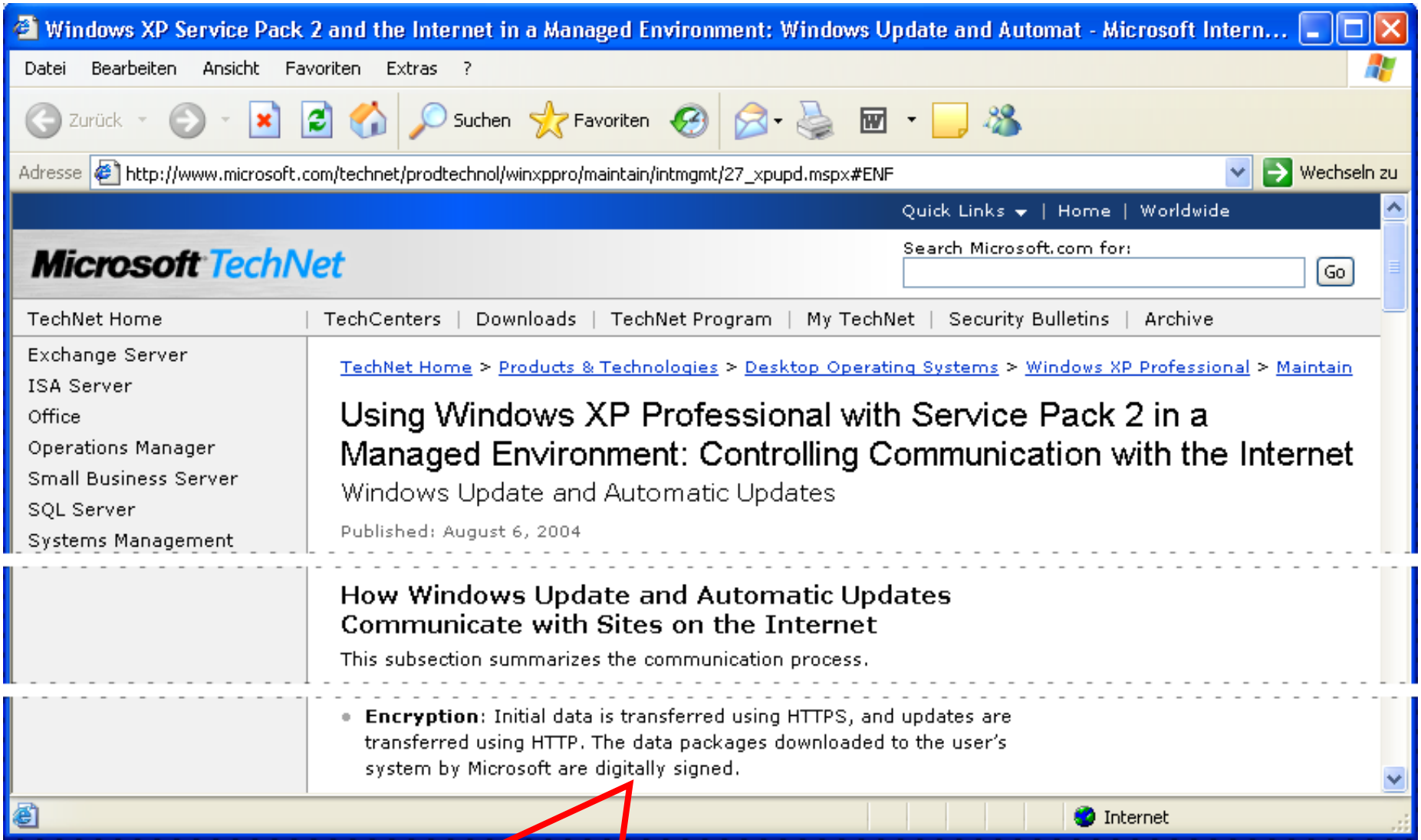
TECHNISCHE
UNIVERSITÄT
DARMSTADT

CMSS – Digital Signatures without Number Theory?

Johannes Buchmann,
Luis Carlos Coronado García,
Erik Dahmen,
Martin Döring,
Elena Klintsevich

Applications of digital signatures



data packages (...) are digitally signed using RSA.

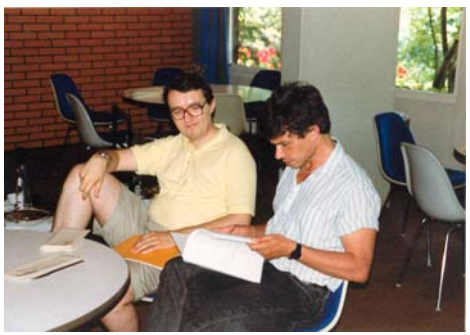
Digitally signed updates

German passports

Personal data ECDSA signed.

2007: ECDSA access control to biometric data

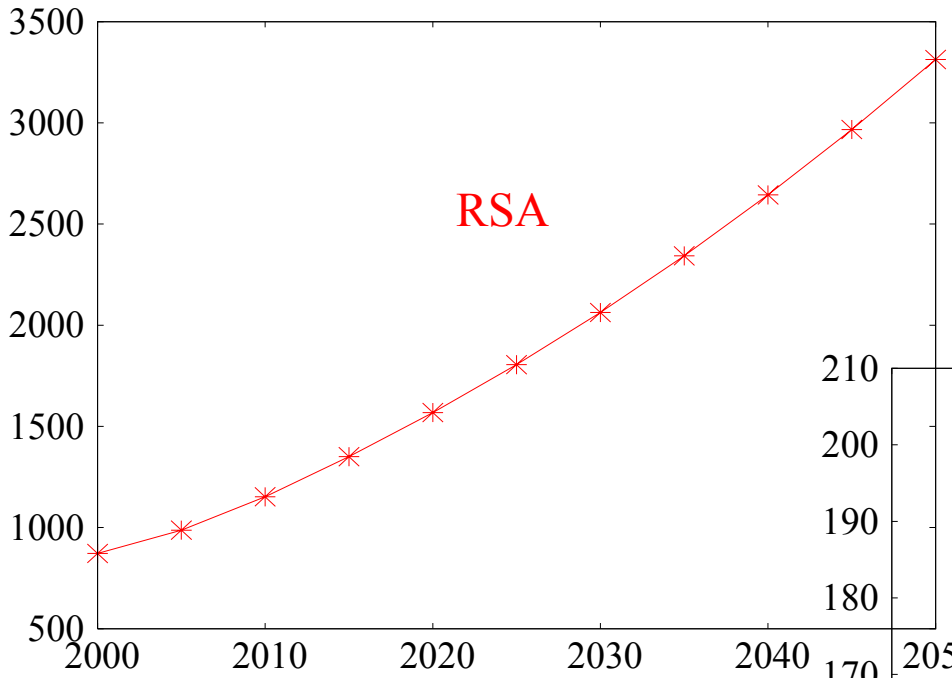
How long will RSA and ECDSA be secure?

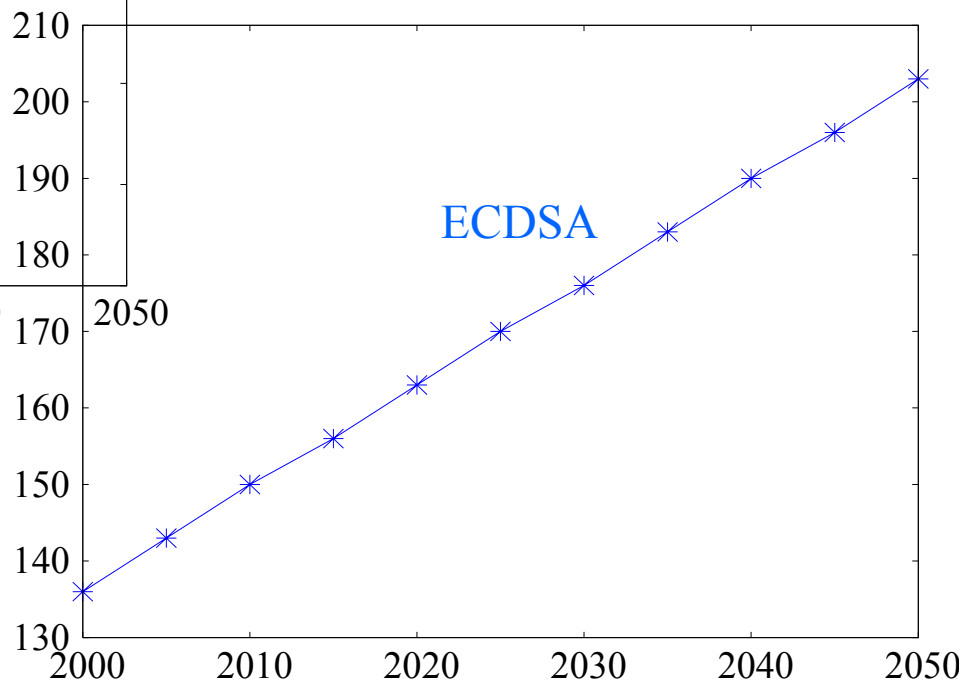




Selecting key sizes

Lenstra (2004)

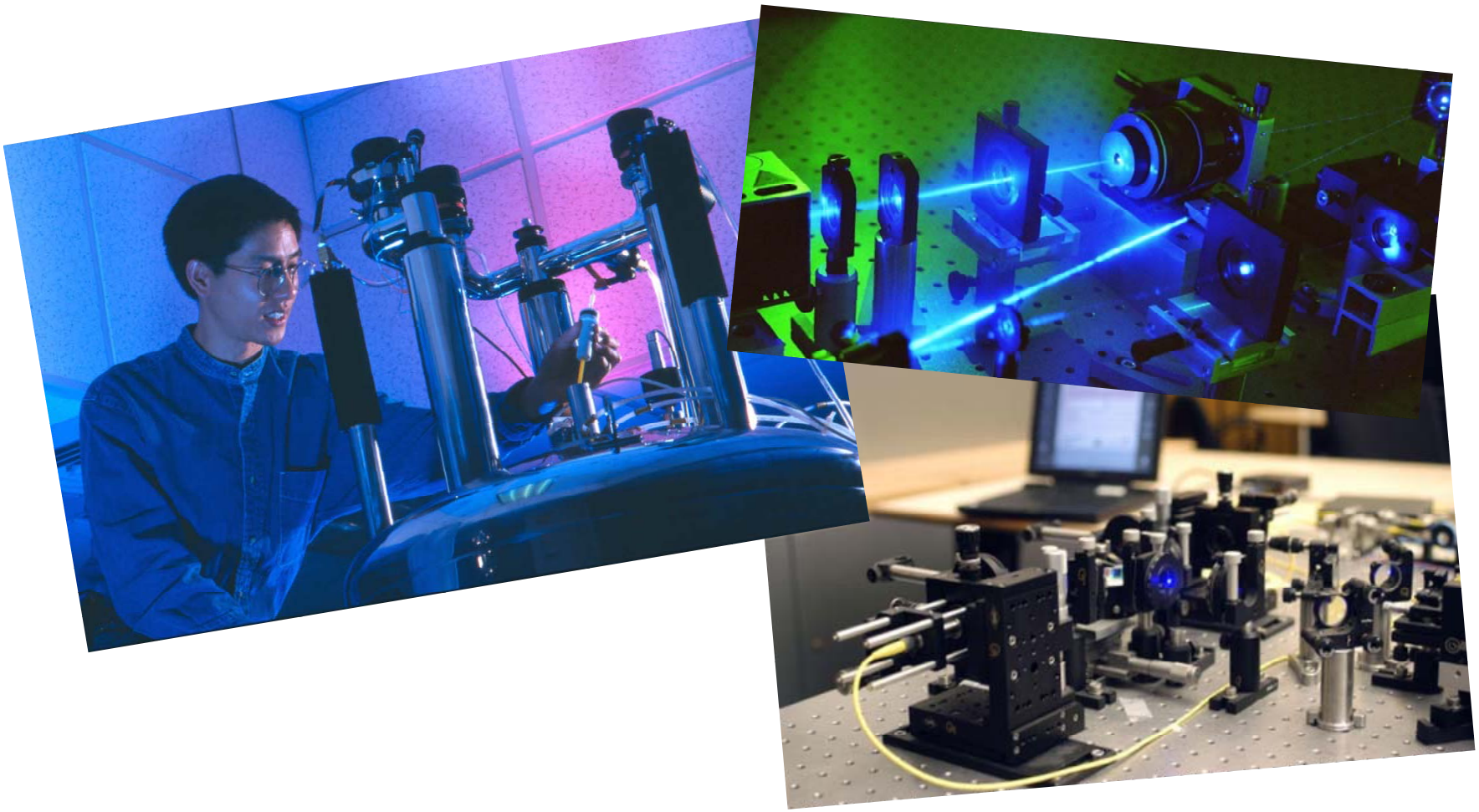




	2006	2046
RSA	1012 bits	3034 bits
ECDSA	144 bits	198 bits

Quantum computers will make RSA, DSA, ECDSA insecure

Shor (1994)



Alternatives?

Rompel (1990)

One-way functions are necessary and sufficient for secure signatures

- Hash functions suffice

Impagliazzo, Rudich (1989)

(Black box) one-way functions are not sufficient for public key encryption

- Number theoretic, algebraic, geometric problems
- required?

A signature scheme based on hash functions

Merkle (1979)

Merkle signature scheme

Idea:

Hash based one-time signature scheme (OTSS)

One key pair ( , ) per signature

Hash tree:

Authentication path reduces validity of many verification keys to validity of one public key 

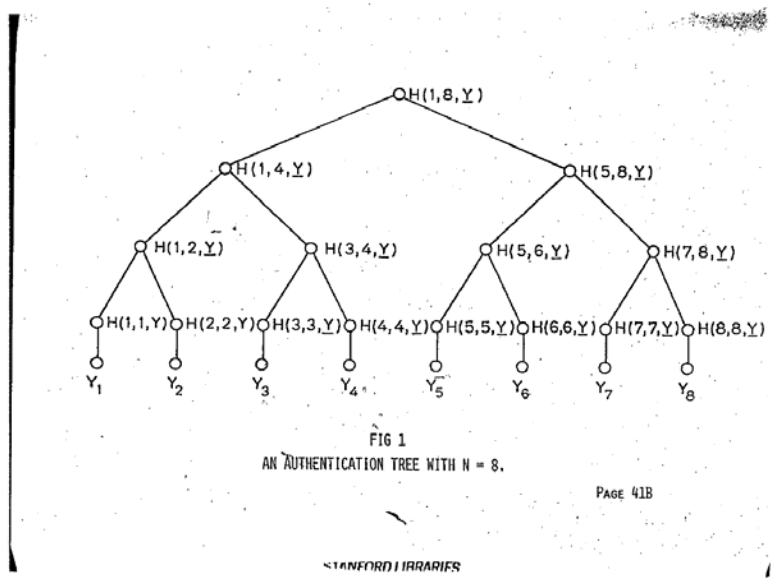


FIG 1
AN AUTHENTICATION TREE WITH N = 8.

PAGE 41B

Lamport-Diffie OTSS

Lamport, Diffie (1976)

System parameters:

Hash function $H: \{0,1\}^* \rightarrow \{0,1\}^n$

Key generation

Signature key

$= x_1(0), x_1(1), x_2(0), x_2(1), \ldots, x_n(0), x_n(1) \in_R \{0,1\}^n$

$\downarrow H$

$= y_1(0), y_1(1), y_2(0), y_2(1), \ldots, y_n(0), y_n(1) \in \{0,1\}^n$

Verification key

Lamport-Diffie OTSS

Example

$n = 3$

$x_1(0), x_1(1), x_2(0), x_2(1), x_3(0), x_3(1)$

0 1 1 0 0 1

1 1 1 0 1 0

0 0 1 1 1 1

H

0 1 0 0 1 1

1 1 0 1 0 1

1 1 0 0 0 0

$y_1(0), y_1(1), y_2(0), y_2(1), y_3(0), y_3(1)$

Lamport-Diffie OTSS

Signing

$= m_1, m_2, \ldots, m_v$

$\downarrow H$

$H($  $) = h_1, h_2, \ldots, h_n$

$\downarrow$ 

$= x_1(h_1), x_2(h_2), \ldots, x_n(h_n)$

Lamport-Diffie OTSS

Example

0 1 1 0 0 1
1 1 1 0 1 0
0 0 1 1 1 1

0 1 0 0 1 1
1 1 0 1 0 1
1 1 0 0 0 0

$= 2010$

H

$H($  $) = 0\ 1\ 0$

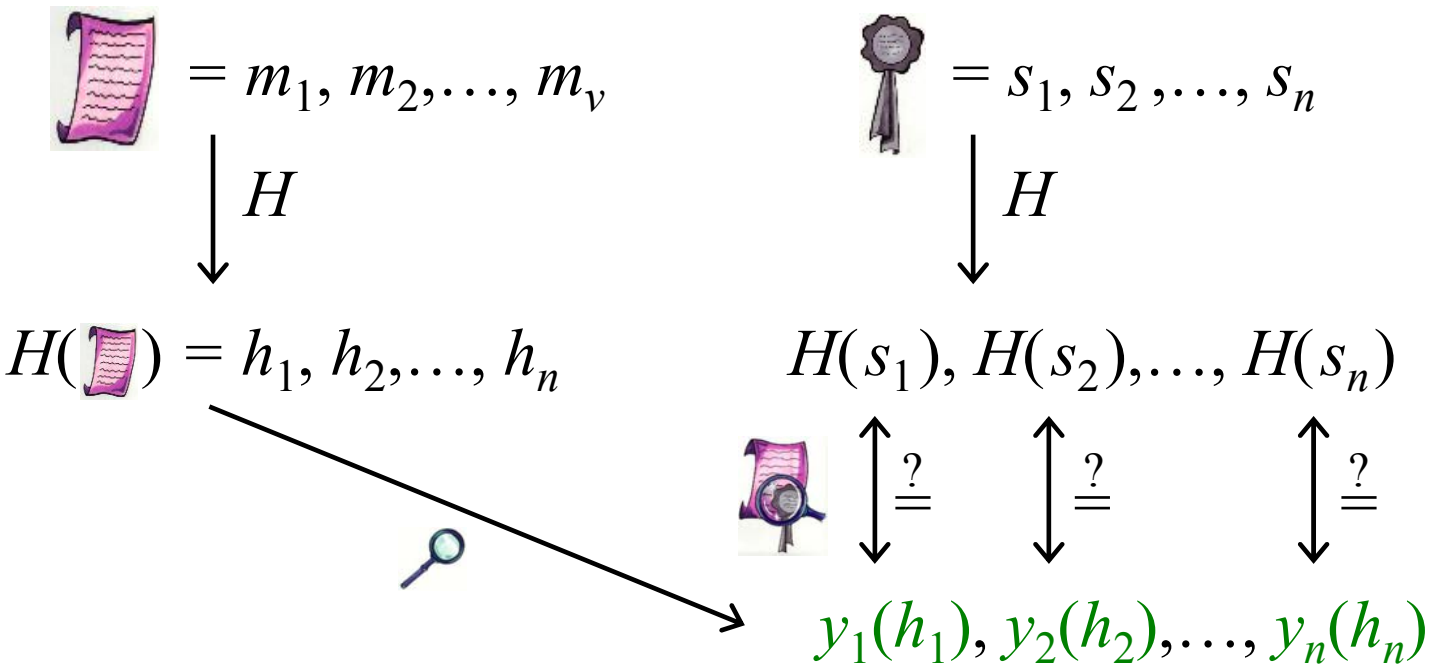
0 0 0

$= 1\ 0\ 1$

$0\ 1\ 1$

Lamport-Diffie OTSS

Verifying



Lamport-Diffie OTSS

Example

0 1 1 0 0 1
1 1 1 0 1 0
0 0 1 1 1 1

0 1 0 0 1 1
1 1 0 1 0 1
1 1 0 0 0 0

 = 2010

H

$H($  $) = 010$

 = 0 0 0
1 0 1
0 1 1

H

0 0 1
1 1 0
1 0 0

 $\stackrel{?}{=}$

0 0 1
1 1 0
1 0 0

Lamport-Diffie OTSS

Example

0 1 1 0 0 1
1 1 1 0 1 0
0 0 1 1 1 1

0 1 0 0 1 1
1 1 0 1 0 1
1 1 0 0 0 0

$= 2006$

H

$H($  $) = 010$

$=$ 0 0 0
1 0 1
0 1 1

H H^{-1}

0 0 1
1 1 0
1 0 0

$\stackrel{?}{=}$

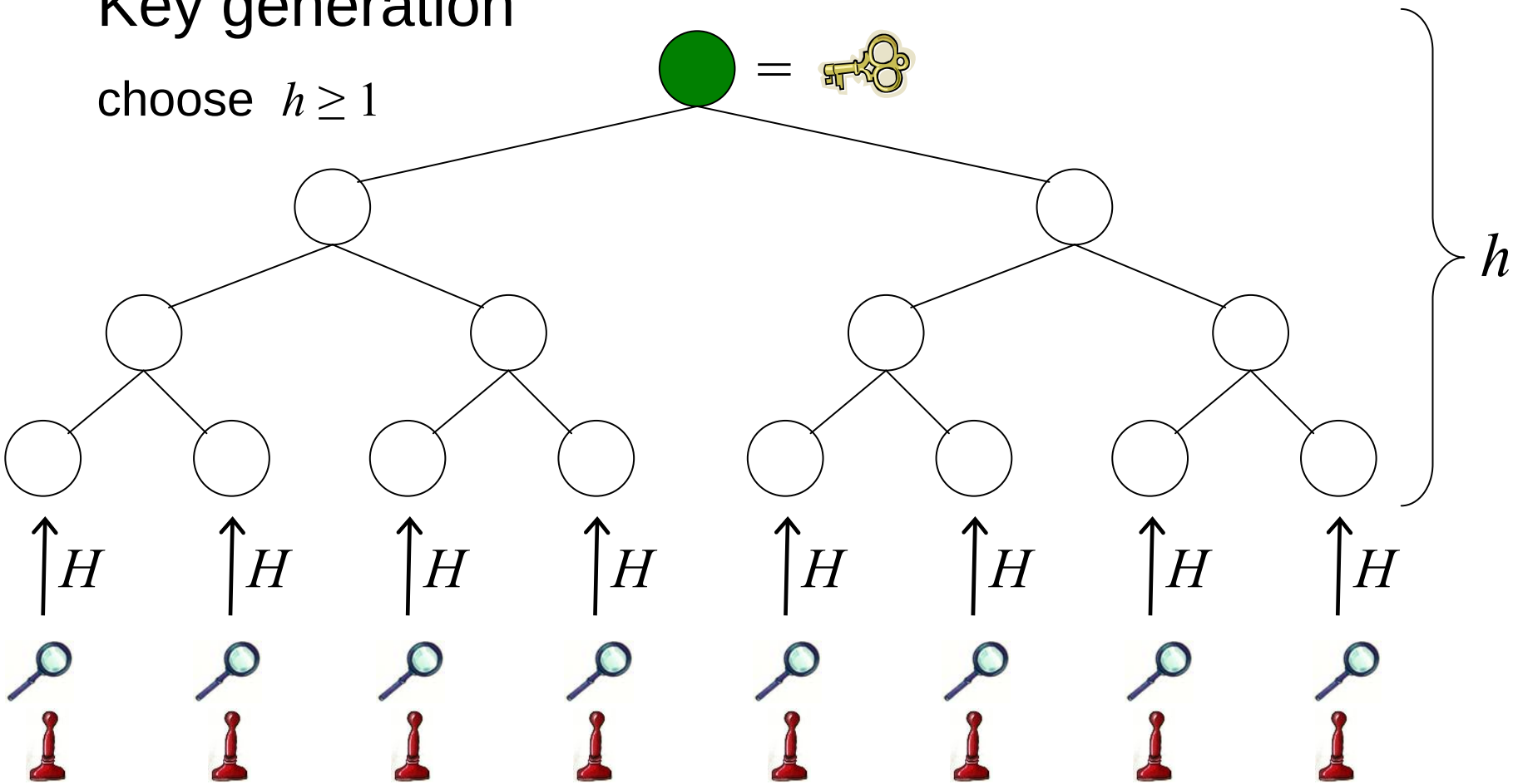
0 0 1
1 1 0
1 0 0

Merkle signature scheme

Merkle (1979)

Key generation

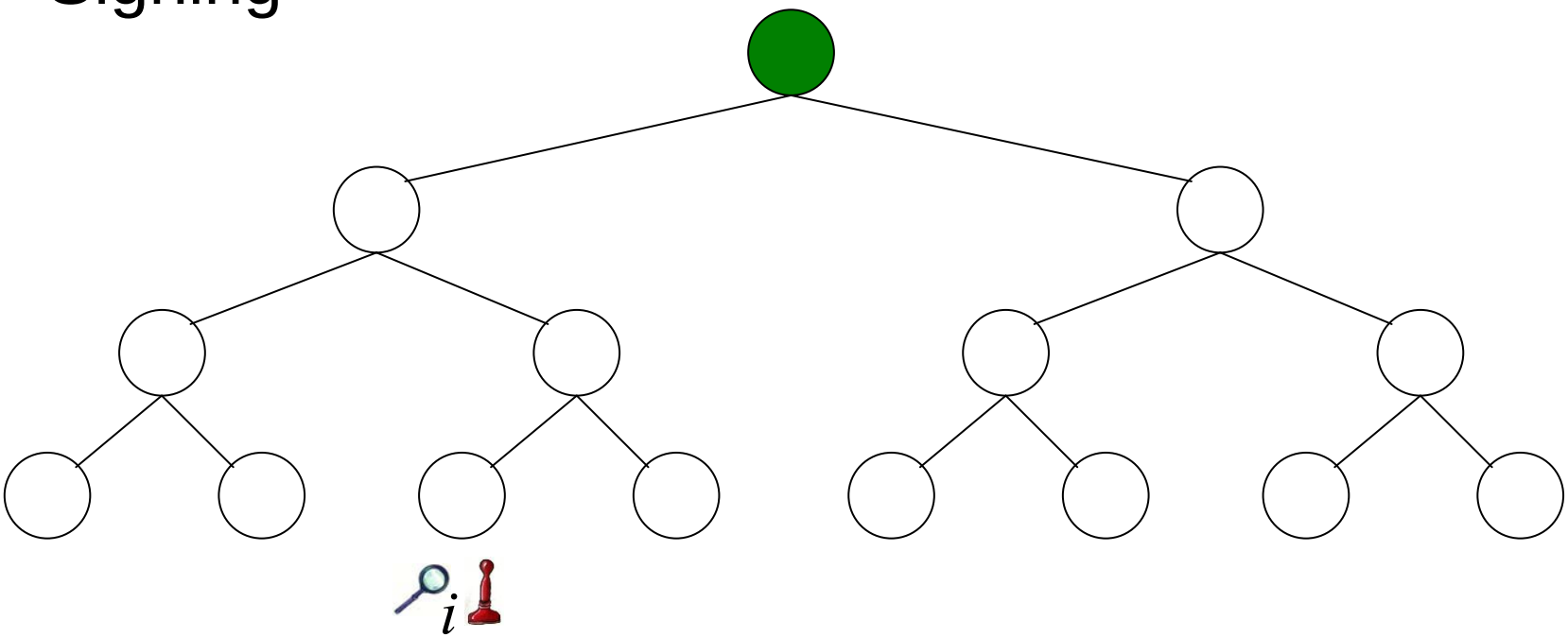
choose $h \geq 1$

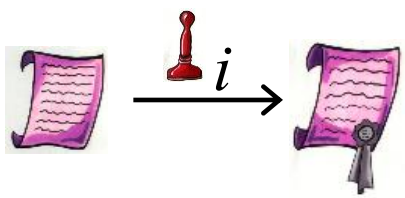


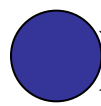
$$parent = H(left \,||\, right)$$

Merkle signature scheme

Signing

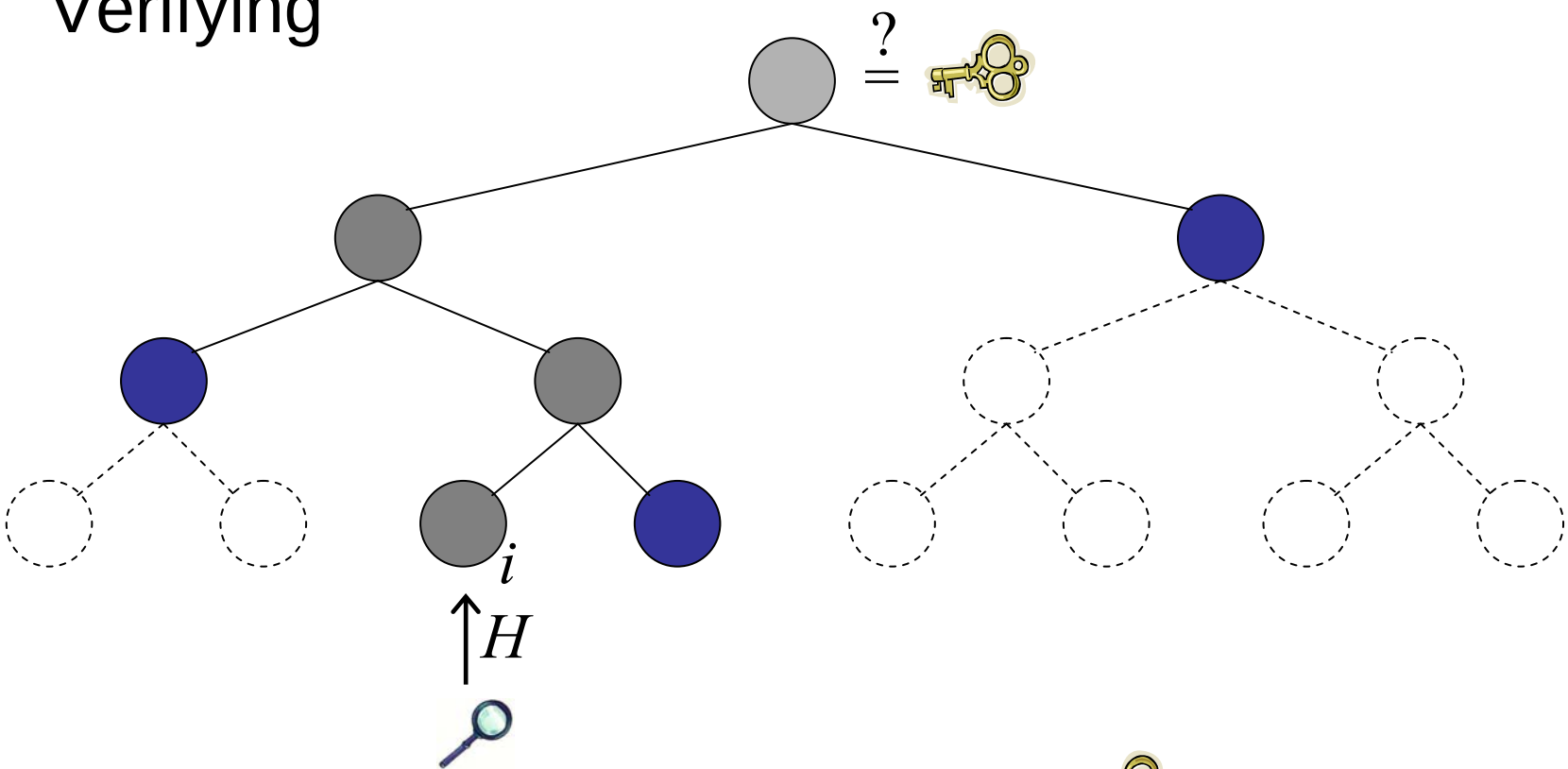


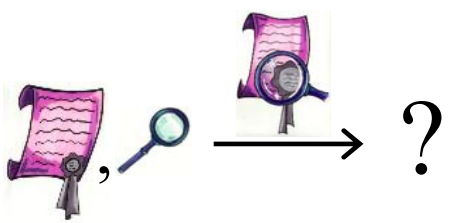


Signature = $(i,$  , ,  ,  ,  $)$

Merkle signature scheme

Verifying





Public key =

Signature = (*i*, , , , ,)

Merkle signature scheme

Problems

- Large memory for private key
- Expensive authentication path computation
- Expensive authentication tree computation

Winternitz OTSS

Winternitz (1979), Dods, Smart, Stam (2005)

System parameters:

Hash function $H: \{0,1\}^* \rightarrow \{0,1\}^n$

$w \geq 1$; number of bits signed simultaneously

$t \approx \left(n + \log(n)\right) / w \quad < 2n$

Signature key

$= x_1, x_2, \ldots, x_t \in_R \{0,1\}^n$

Verification key

$= H(H^{2^w-1}(x_1) \,||\, H^{2^w-1}(x_2) \,||\, \ldots \,||\, H^{2^w-1}(x_t))$

Winternitz OTSS

Signing:

 $= x_1, x_2, \ldots, x_t$

$H($  $) = 100101010\ldots110 \quad 111 \quad \ldots \quad 001$

$b_1 \quad b_2 \quad b_3 \quad \ldots \quad b_{\lceil n/w \rceil} \quad b_{p+1} \quad \ldots \quad b_t$

$\lceil n/w \rceil = p$

$$c = \sum_{i=1}^{p} 2^w - b_i$$

$$(\sigma_1, \sigma_2, \ldots, \sigma_t) = (H^{b1}(x_1), H^{b2}(x_2), \ldots, H^{bt}(x_t))$$

Winternitz OTSS

Verifying:

$$= H(H^{2^w-1}(x_1) \,\|\, H^{2^w-1}(x_2) \,\|\, \ldots \,\|\, H^{2^w-1}(x_t))$$

$$(\sigma_1, \sigma_2, \ldots, \sigma_t) = (H^{b1}(x_1), H^{b2}(x_2), \ldots, H^{bt}(x_t))$$

Compute $b_1, \ldots, b_t$ as before

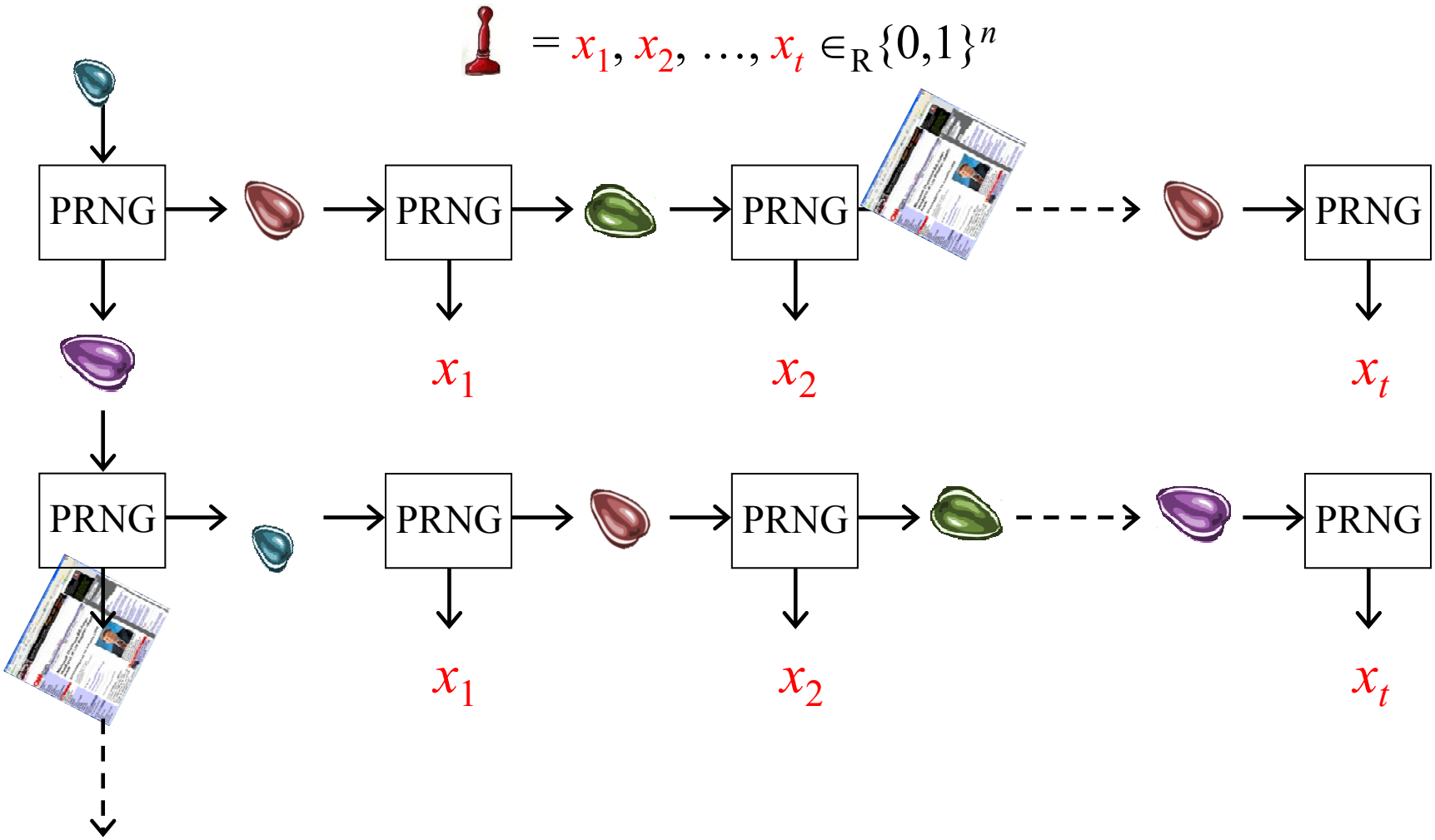
$$(\varphi_1, \varphi_2, \ldots, \varphi_t) = H^{2^w-1-b1}(\sigma_1), H^{2^w-1-b2}(\sigma_2), \ldots, H^{2^w-1-bt}(\sigma_t)$$

$$\stackrel{?}{=} H(\varphi_1 \,\|\, \varphi_2 \,\|\, \ldots \,\|\, \varphi_t)$$

Improved OTSS key generation

Coronado (2005)

truly random



Authentication path computation

Naive:

Store complete authentication tree: 2^{h+1} nodes

Szydlo (2004):

Suffices to store $3h$ nodes and compute $2h$ hash values per signature

Authentication path computation

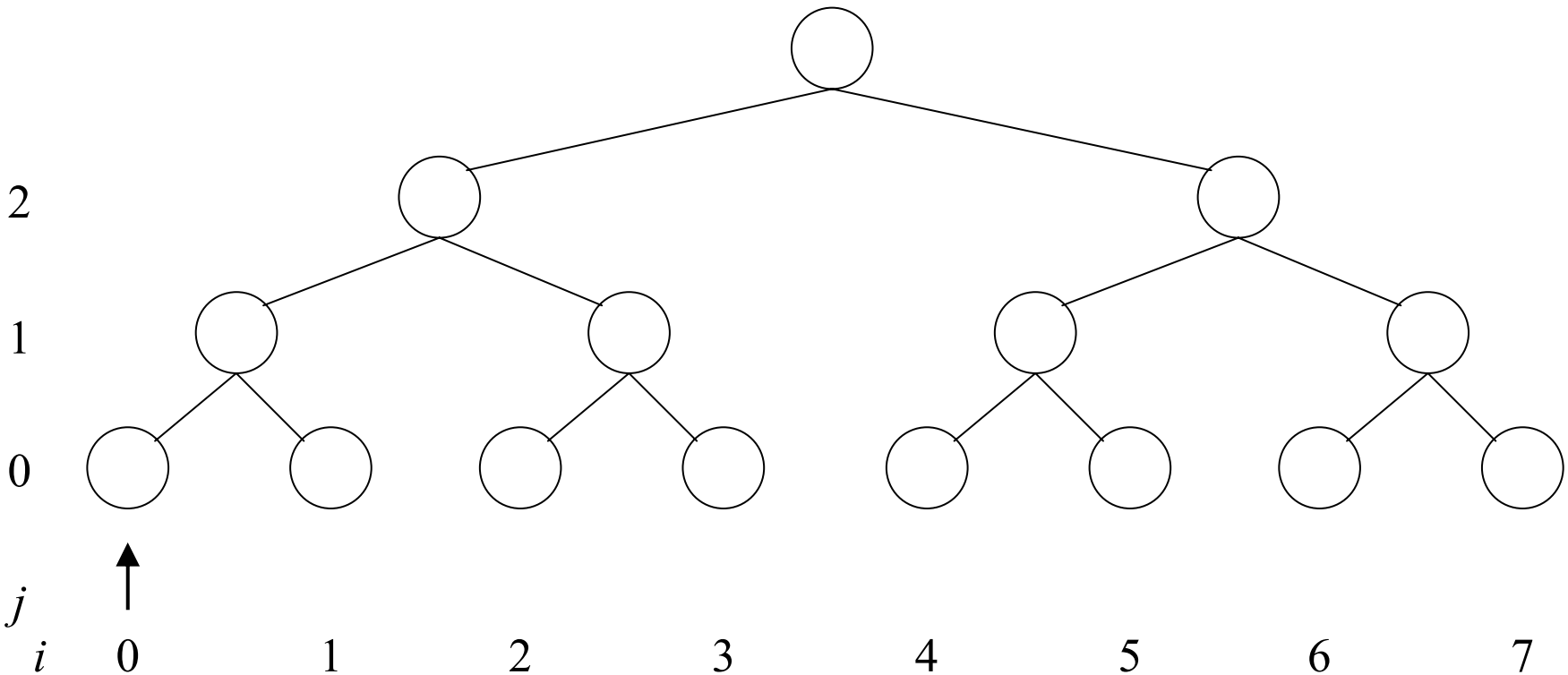
Replace node j if 2^j divides $i+1$

Initialize and Update stacks

$$startnode = (i + 1 + 2^j) \oplus 2^j$$

$2^0 \mid i+1=1$

$(1+ 2^0) \oplus 2^0 = 3$



Authentication path computation

Replace node j if 2^j divides $i+1$

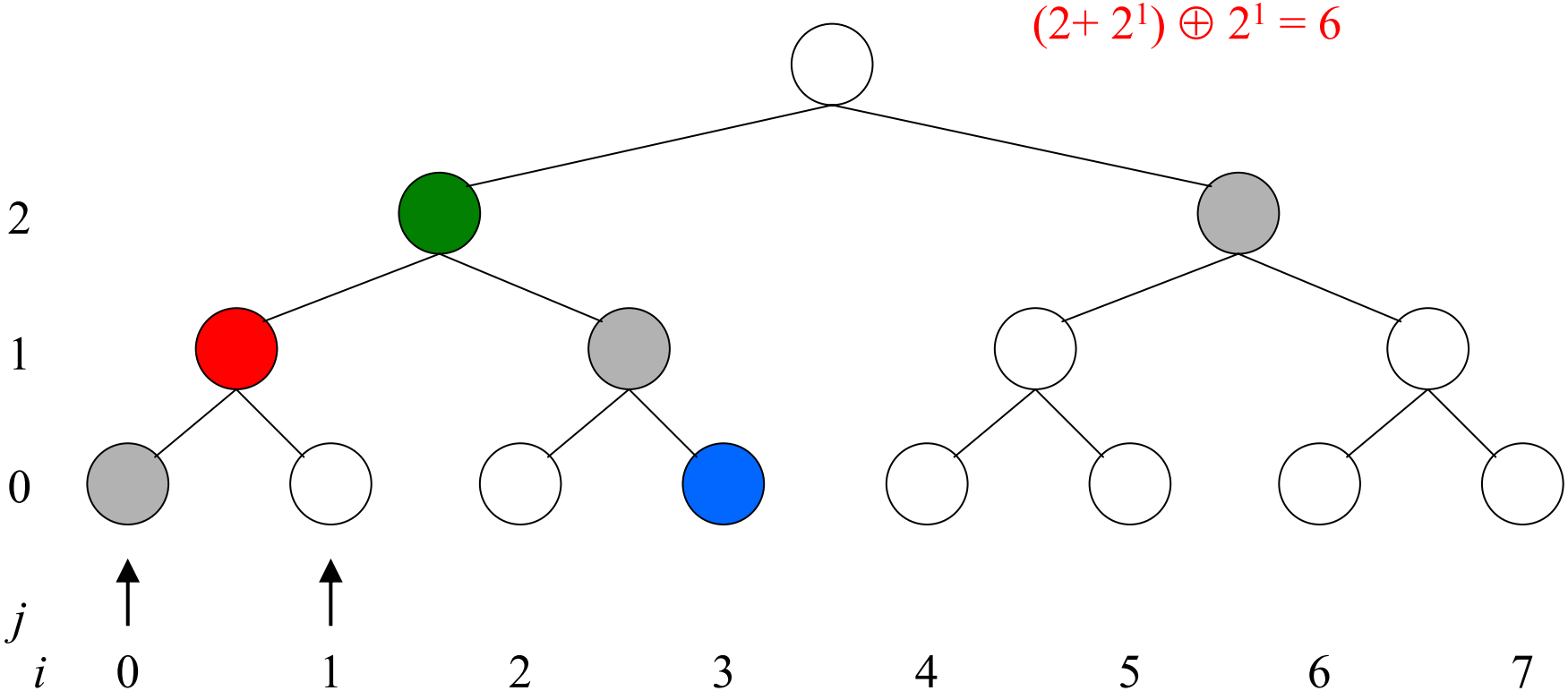
Initialize and Update stacks

$$startnode = (i + 1 + 2^j) \oplus 2^j$$

$2^0 \mid i+1=2$ $\quad$ $2^1 \mid 2$

$(2+ 2^0) \oplus 2^0 = 2$

$(2+ 2^1) \oplus 2^1 = 6$



Authentication path computation

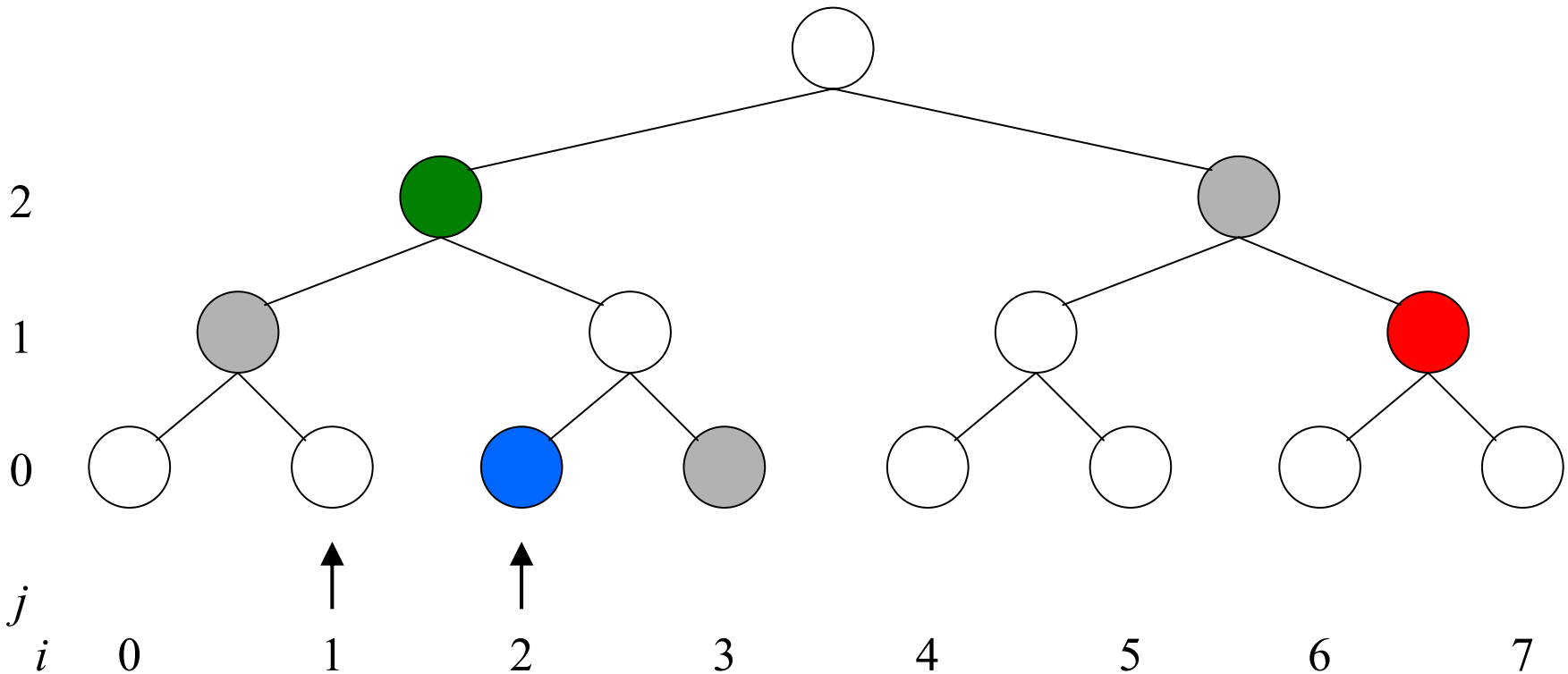
Replace node j if 2^j divides $i+1$

Initialize and Update stacks

$$startnode = (i + 1 + 2^j) \oplus 2^j$$

$2^0 \mid i+1=3$

$(3+ 2^0) \oplus 2^0 = 5$



Authentication path computation

Replace node j if 2^j divides $i+1$

Initialize and Update stacks

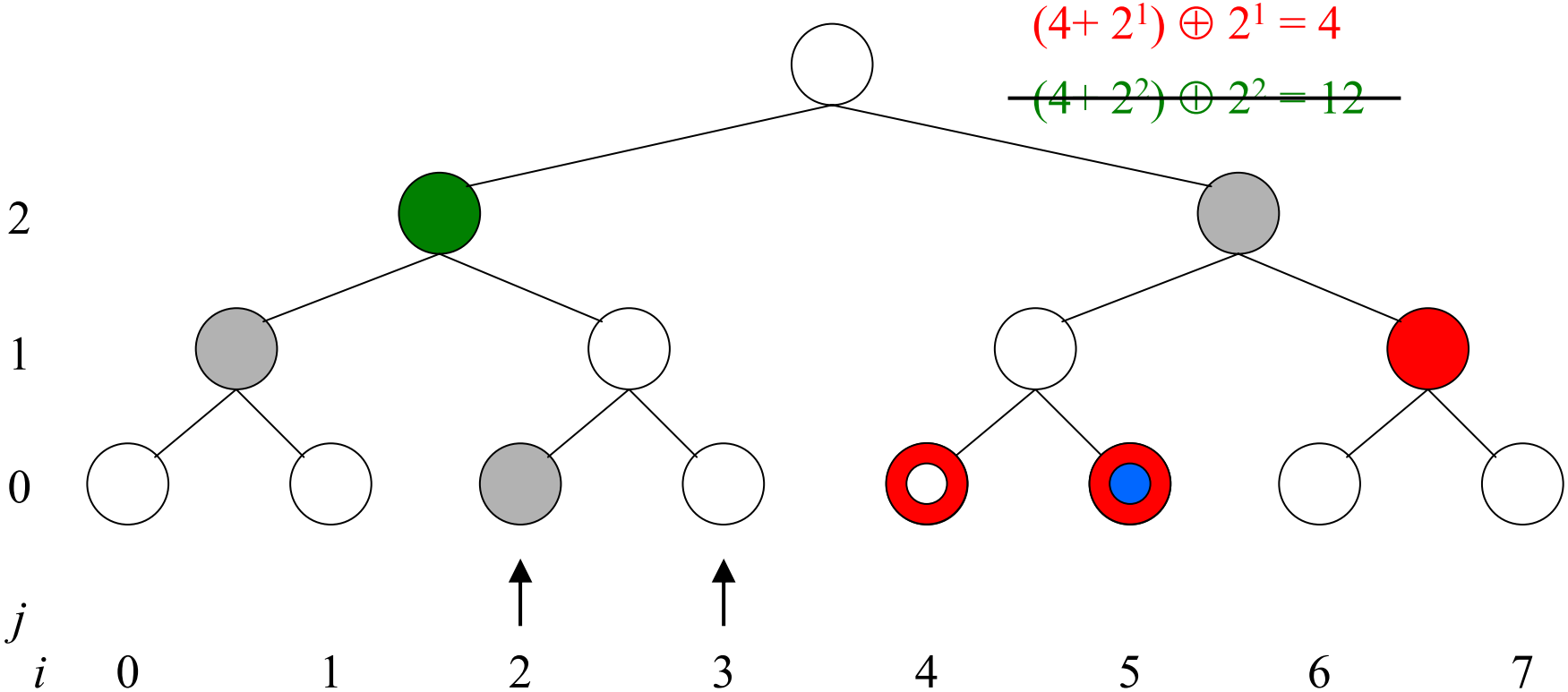
$$startnode = (i + 1 + 2^j) \oplus 2^j$$

$2^0 \mid i+1=4$ $\quad 2^1 \mid 4$ $\quad 2^2 \mid 4$

$(4+ 2^0) \oplus 2^0 = 4$

$(4+ 2^1) \oplus 2^1 = 4$

~~$(4+ 2^2) \oplus 2^2 = 12$~~



Authentication path computation

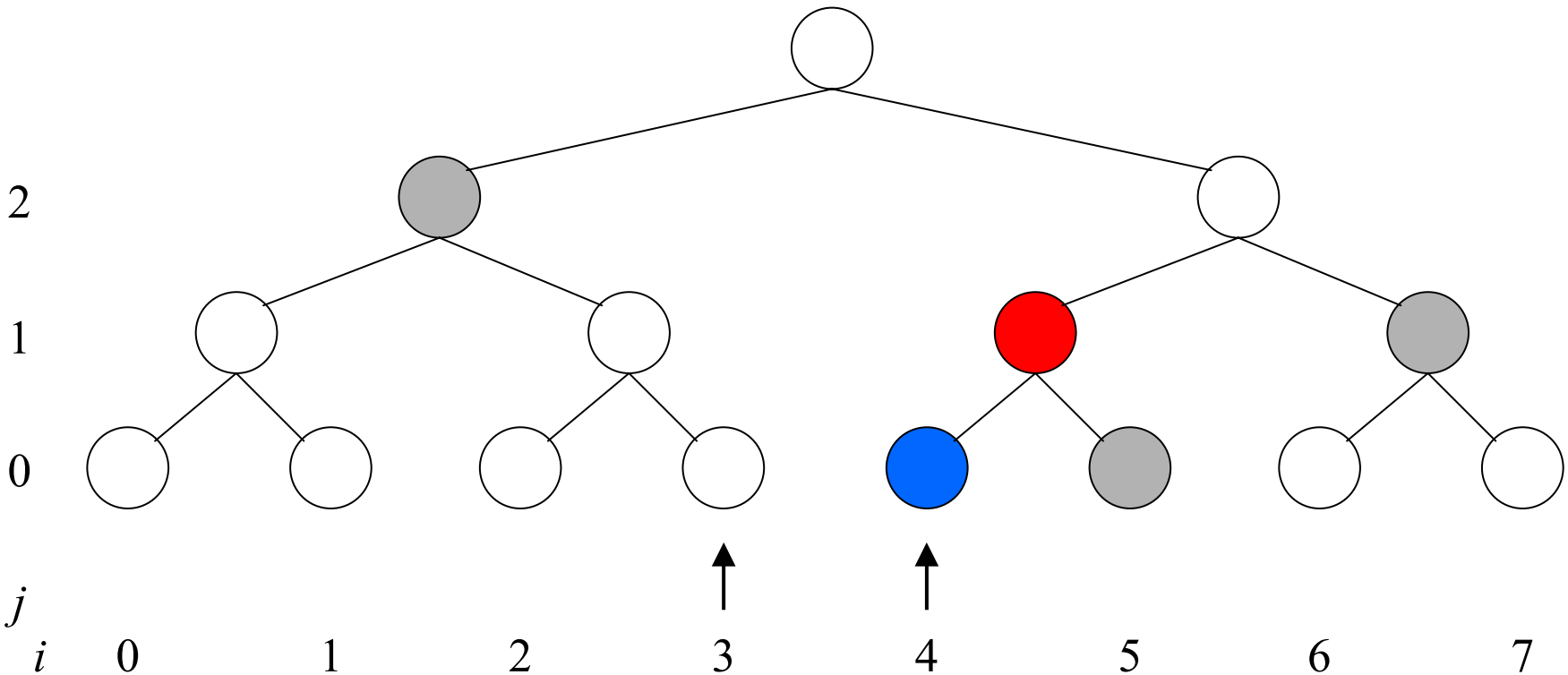
Replace node j if 2^j divides $i+1$

Initialize and Update stacks

$$startnode = (i + 1 + 2^j) \oplus 2^j$$

$2^0 \mid i+1=5$

$(5+ 2^0) \oplus 2^0 = 7$



Authentication path computation

Replace node j if 2^j divides $i+1$

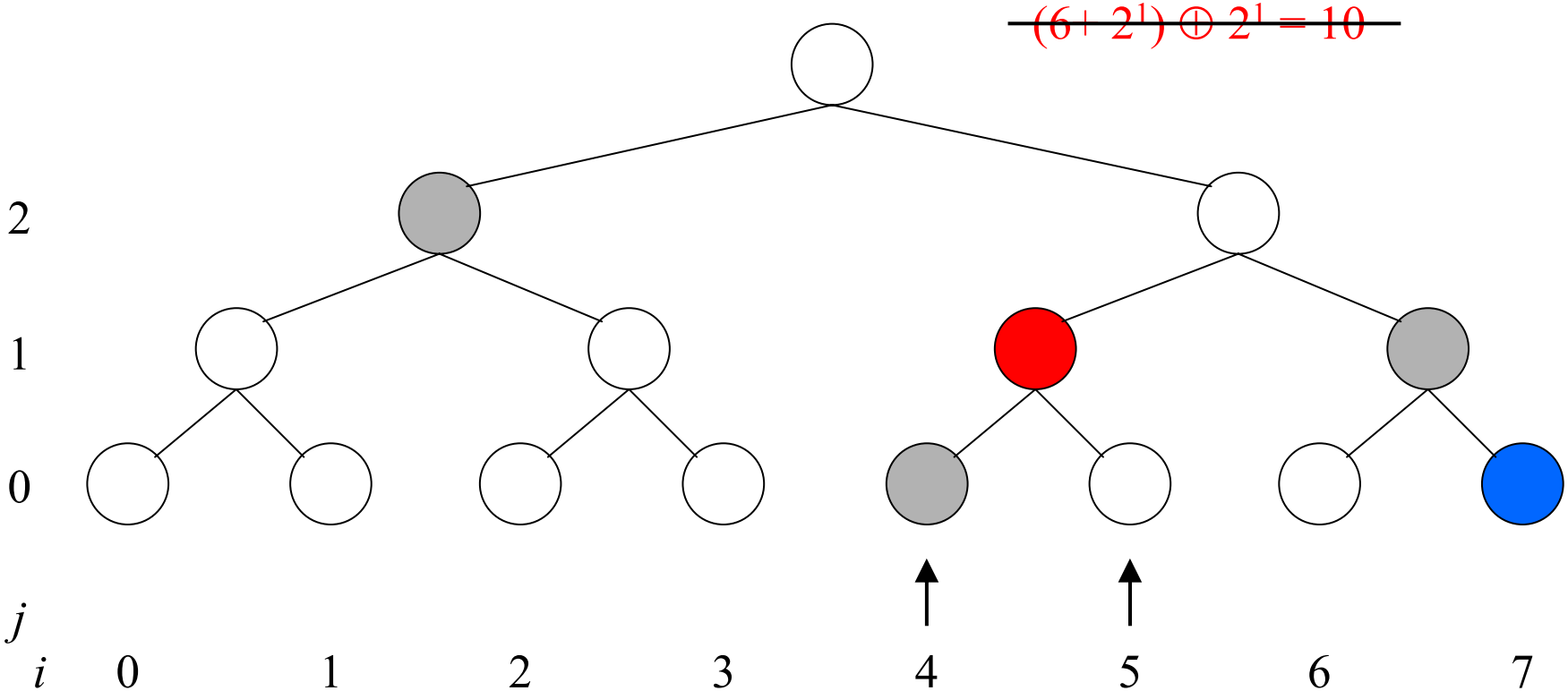
Initialize and Update stacks

$$startnode = (i + 1 + 2^j) \oplus 2^j$$

$2^0 \mid i+1=6 \qquad 2^1 \mid 6$

$(6+ 2^0) \oplus 2^0 = 6$

~~$(6+ 2^1) \oplus 2^1 = 10$~~



Authentication path computation

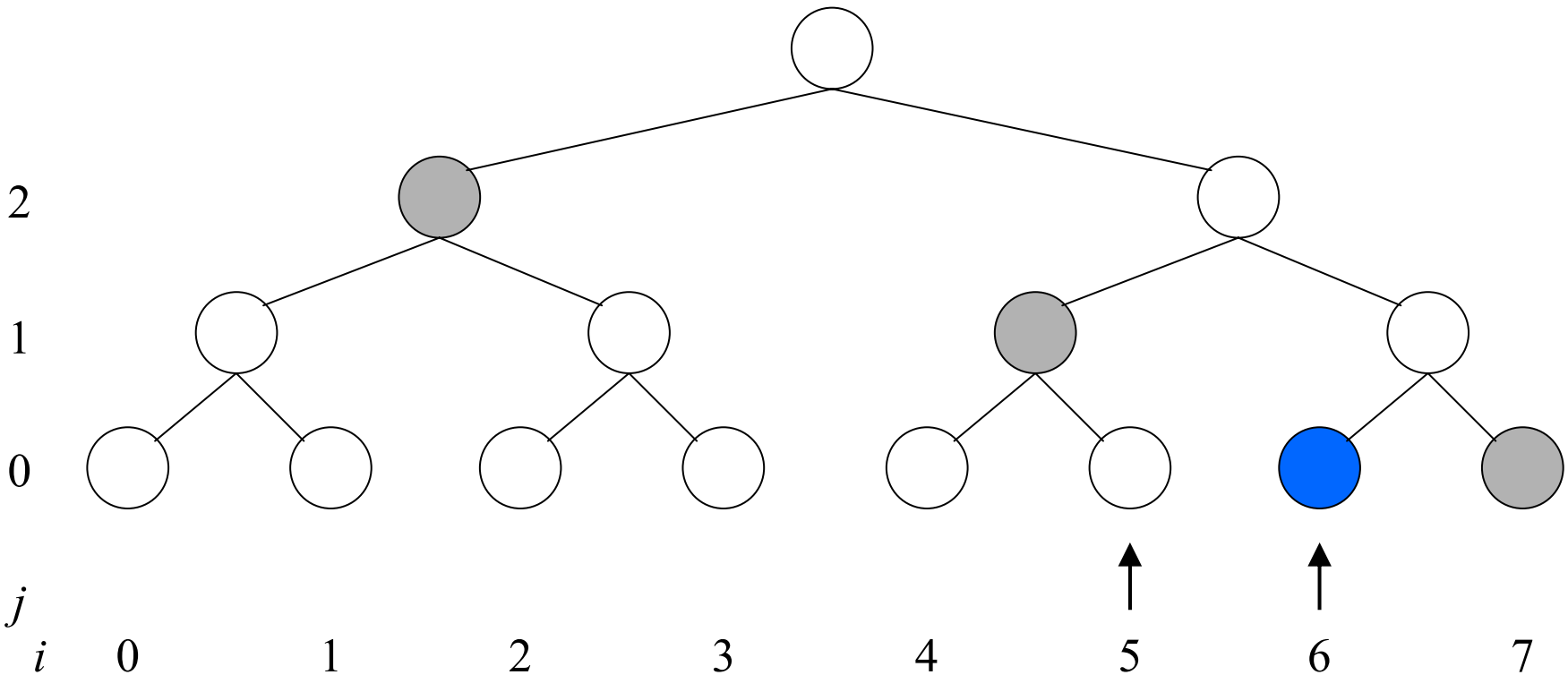
Replace node j if 2^j divides $i+1$

$2^0 \mid i+1=7$

Initialize and Update stacks

$startnode = (i + 1 + 2^j) \oplus 2^j$

~~$(7+2^0) \oplus 2^0 = 9$~~

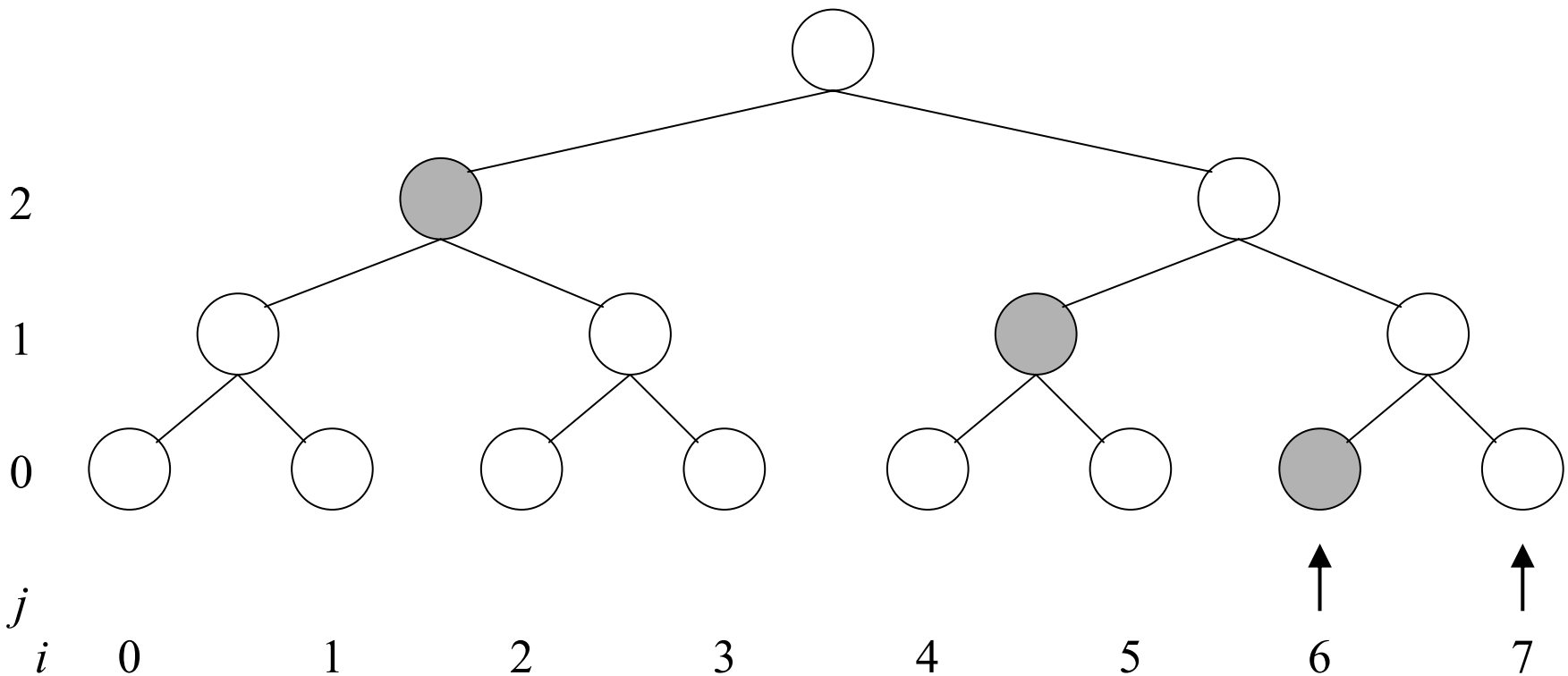


Authentication path computation

Replace node j if 2^j divides $i+1$

Initialize and Update stacks

$$startnode = (i + 1 + 2^j) \oplus 2^j$$



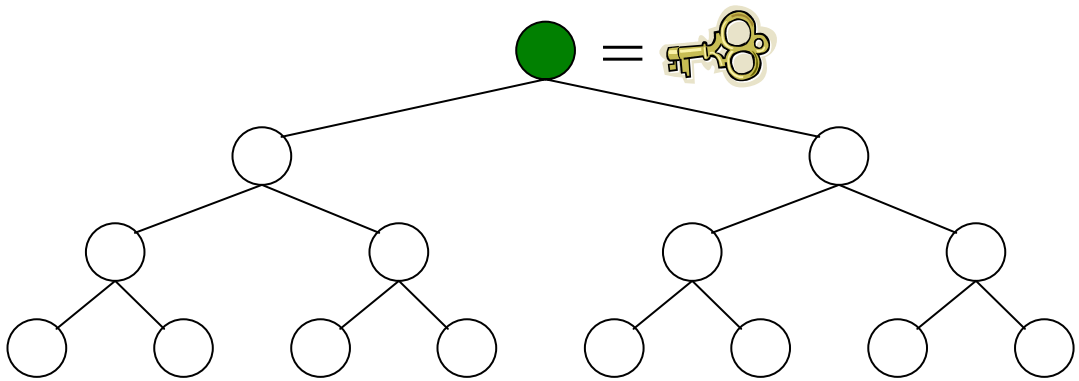
Extrapolated timings for key generation

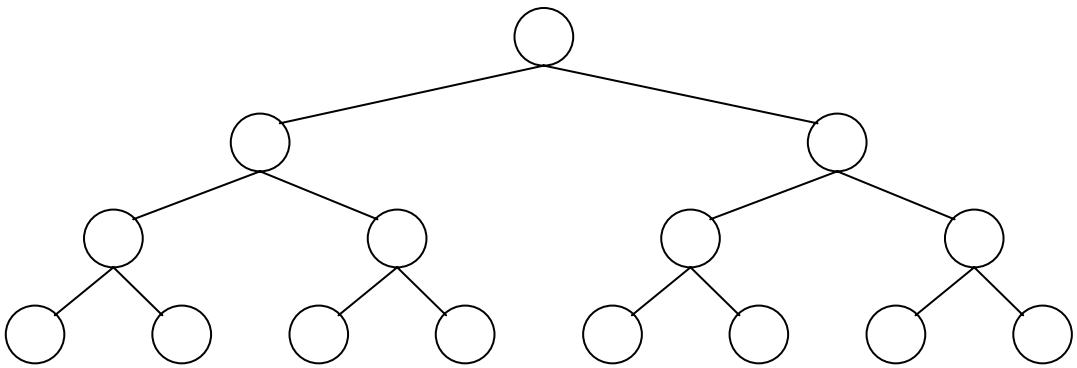
Based on results by Naor, Shenhavy, Woolz (2005)

h	Key generation
20	2.7 min
25	1.4 h
30	45.8 h
35	61.6 days
40	5.4 years

Tree chaining

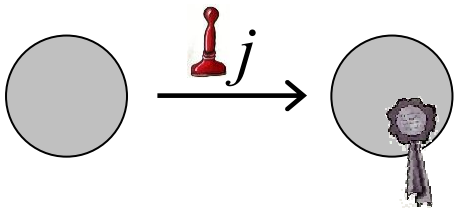
Key generation

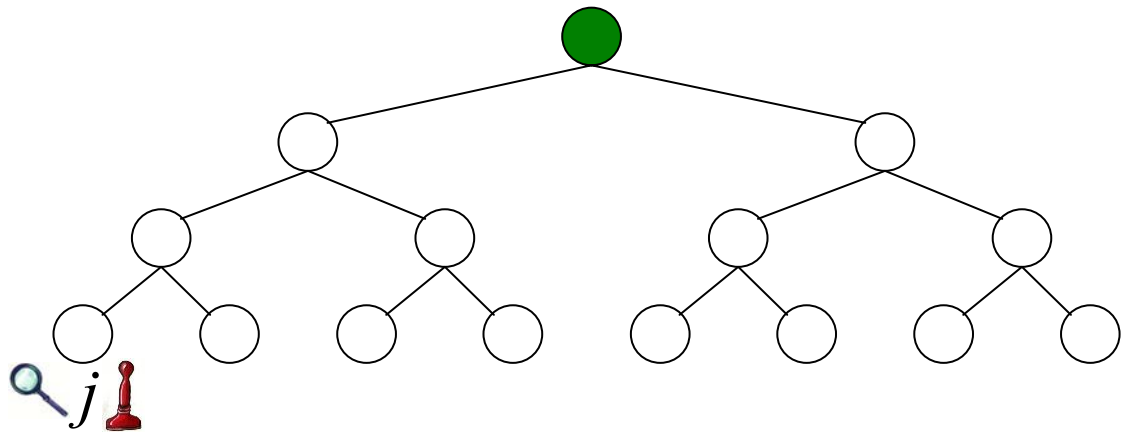


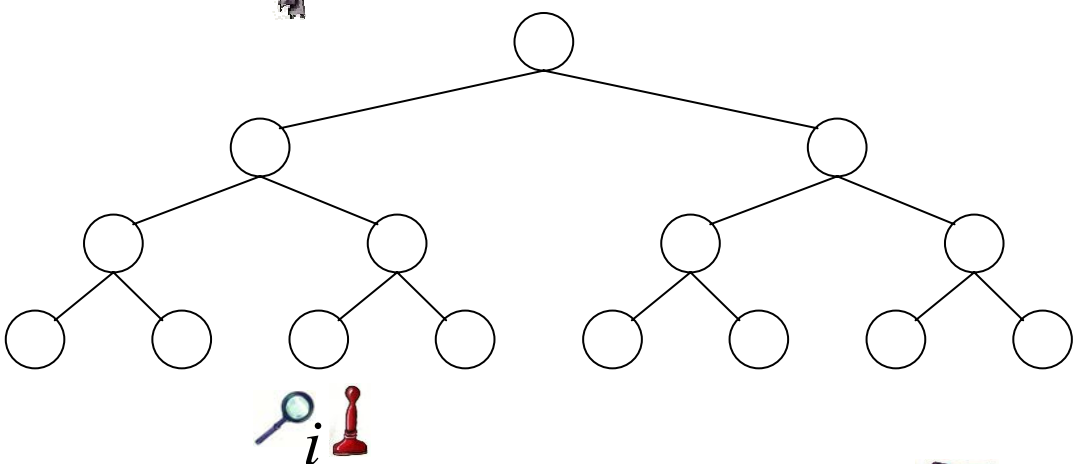


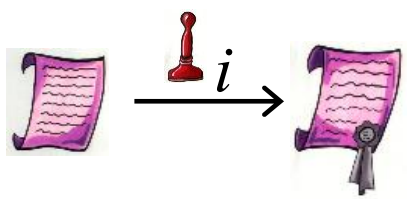
Tree chaining

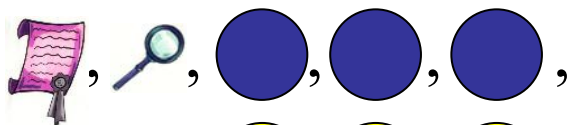
Signing

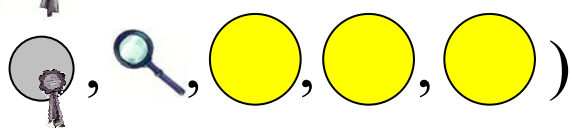




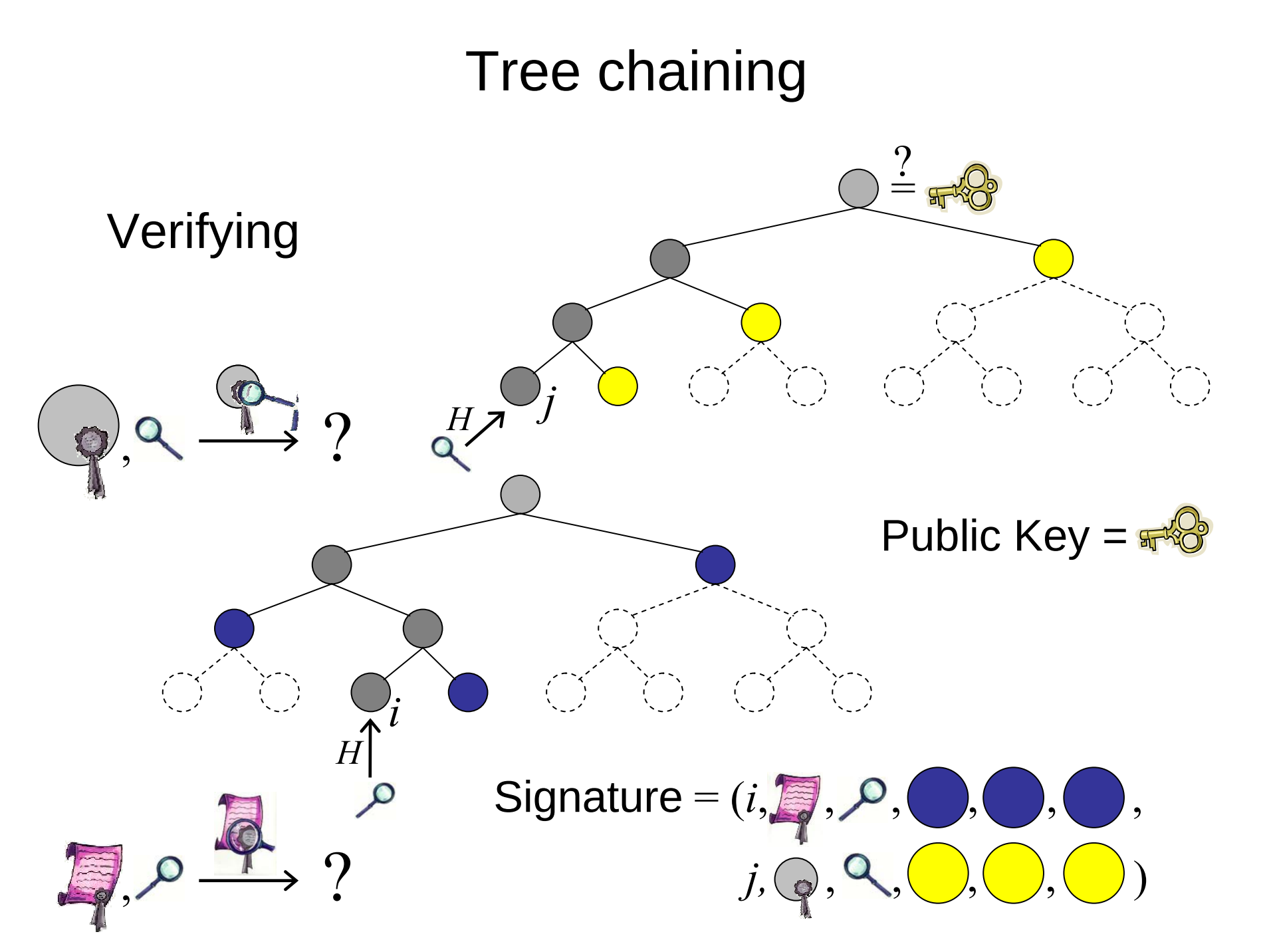




Signature = $(i,$  $,$

$j,$  $)$

Tree chaining



Timings using SHA1

$w = 2$

h	Key generation	Signing	Verifying
20	2.6 sec	9.2 msec	1.3 msec
30	1.4 min	12.4 msec	1.4 msec
40	43.8 min	14.9 msec	1.3 msec

$w = 3$

h	Key generation	Signing	Verifying
20	3.1 sec	9.7 msec	1.5 msec
30	1.5 min	13.2 msec	1.5 msec
40	47.8 min	16.9 msec	1.6 msec

h corresponds to 2^h possible signatures

Sizes using SHA1

$w = 2$

h	Public Key	Private Key	Signature
20	46 bytes	1900 bytes	3808 bytes
30	46 bytes	2788 bytes	4008 bytes
40	46 bytes	3668 bytes	4208 bytes

$w = 3$

h	Public Key	Private Key	Signature
20	46 bytes	1900 bytes	2688 bytes
30	46 bytes	2788 bytes	2888 bytes
40	46 bytes	3668 bytes	3088 bytes

h corresponds to 2^h possible signatures

Timings using SHA256

$w = 2$

h	Key generation	Signing	Verifying
20	6.3 sec	19.6 msec	2.8 msec
30	3.2 min	27.3 msec	2.8 msec
40	101.3 min	34.9 msec	2.9 msec

$w = 3$

h	Key generation	Signing	Verifying
20	7.5 sec	23.3 msec	3.7 msec
30	3.8 min	31.9 msec	3.7 msec
40	120.7 min	40.9 msec	3.7 msec

h corresponds to 2^h possible signatures

Sizes using SHA256

$w = 2$

h	Public Key	Private Key	Signature
20	58 bytes	2884 bytes	9160 bytes
30	58 bytes	4244 bytes	9480 bytes
40	58 bytes	5604 bytes	9800 bytes

$w = 3$

h	Public Key	Private Key	Signature
20	58 bytes	2884 bytes	6408 bytes
30	58 bytes	4244 bytes	6728 bytes
40	58 bytes	5604 bytes	7048 bytes

h corresponds to 2^h possible signatures

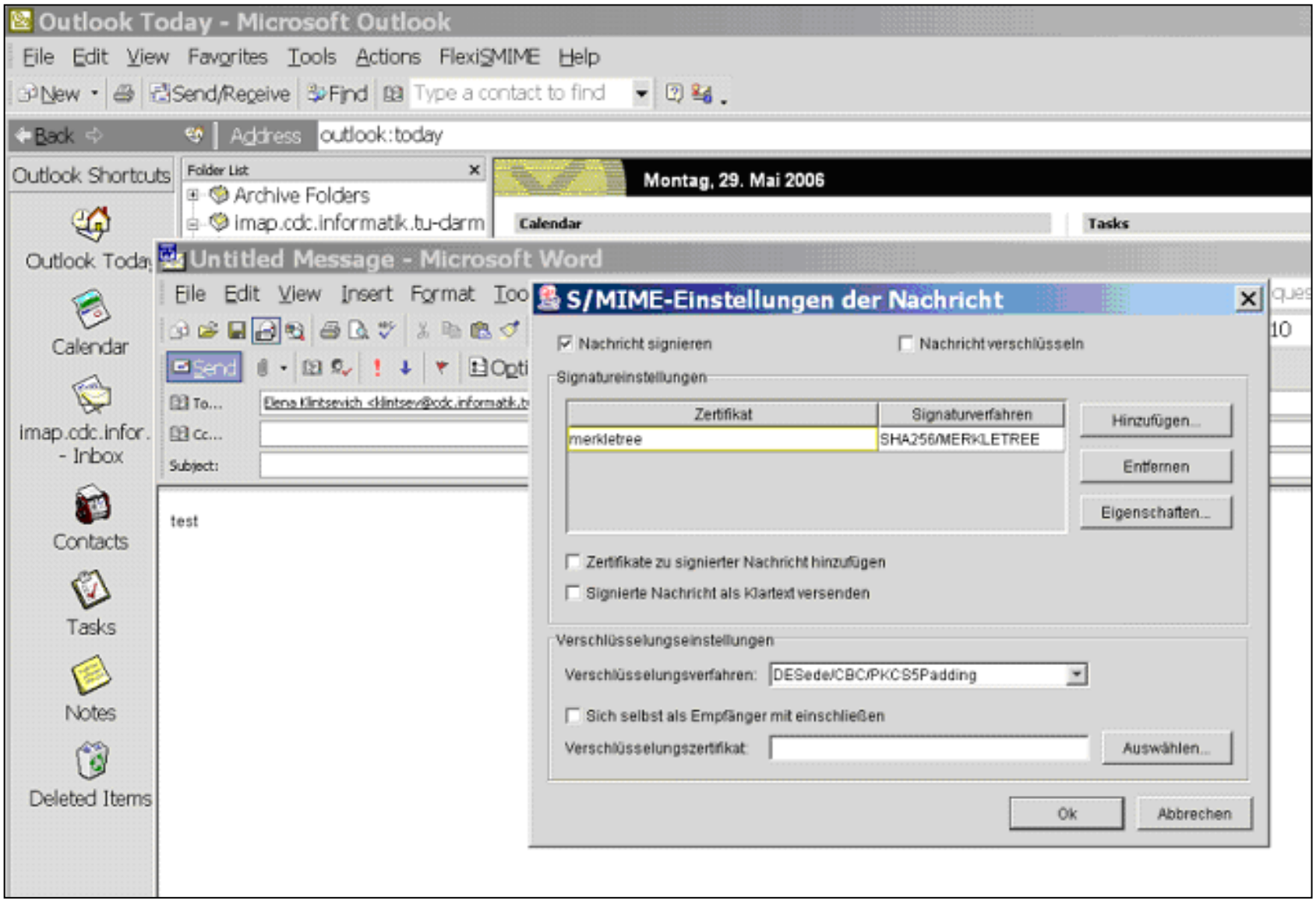
Number theory revisited

Ajtai, Micciancio, Regev, Lyubashevsky Peikert, Rosen

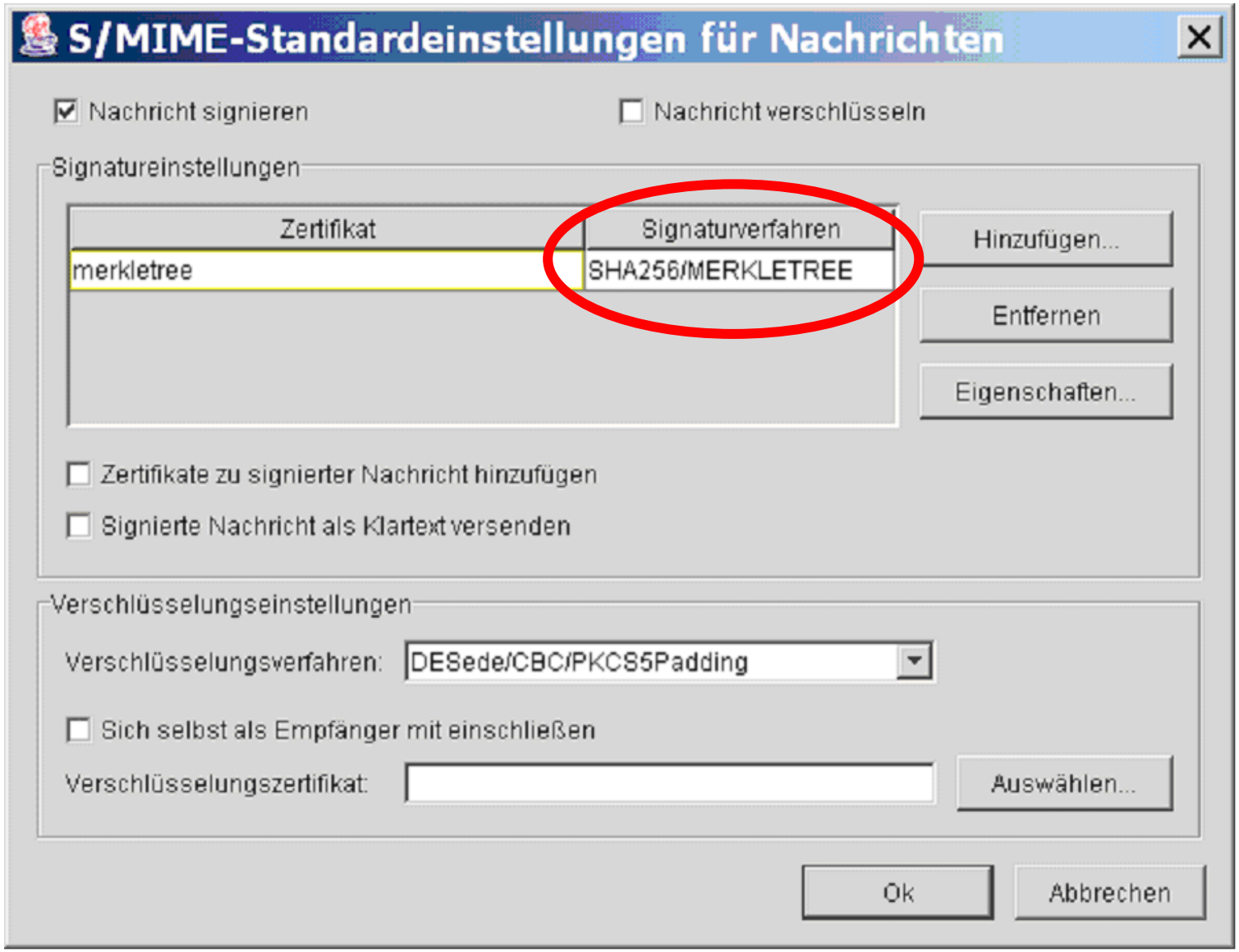
Provably secure hash functions based on lattice reduction

→Provably secure and efficient (?) signature schemes based on lattices

Plugin for MS Outlook



Using Merkle signature scheme

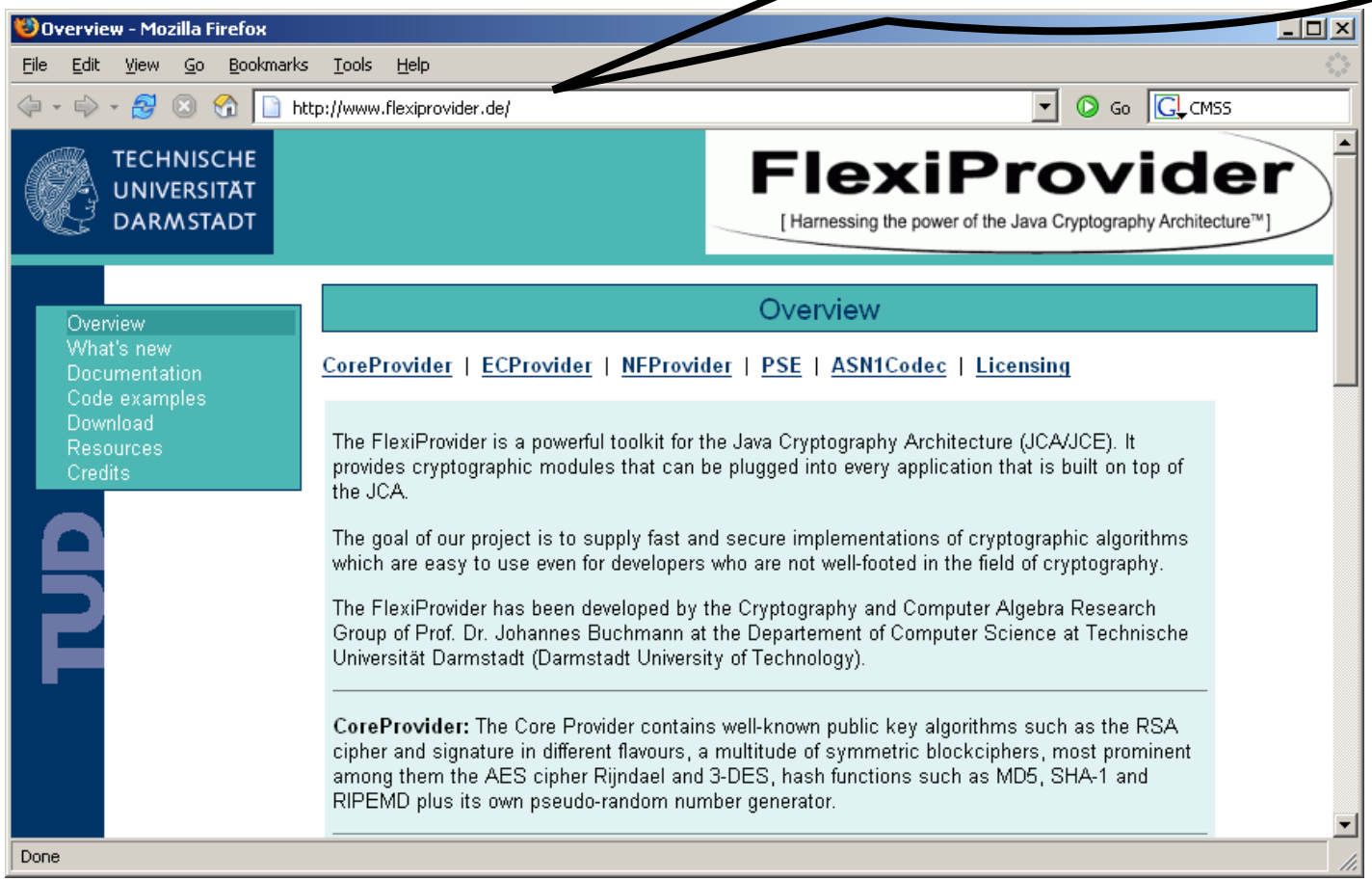


FlexiProvider

Collection of implementations of cryptographic algorithms

Supports CMSS

www.flexiprovider.de



Thank you!