# NIST's Digital Signature Proposal

## A Technical Review

**Burt Kaliski**
**RSA Laboratories**

**Dennis Branstad**
**NIST**

1993 RSA Data Security Conference

Copyright © 1993 RSA Laboratories. All rights reserved.

901-511002-100-000-000

# Outline

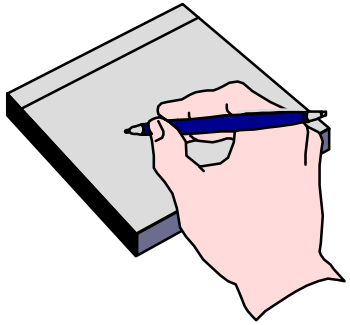
- **Introduction**
- **Definition**
- **Performance**
- **Security**
- **Trap Doors**
- **Conclusions**

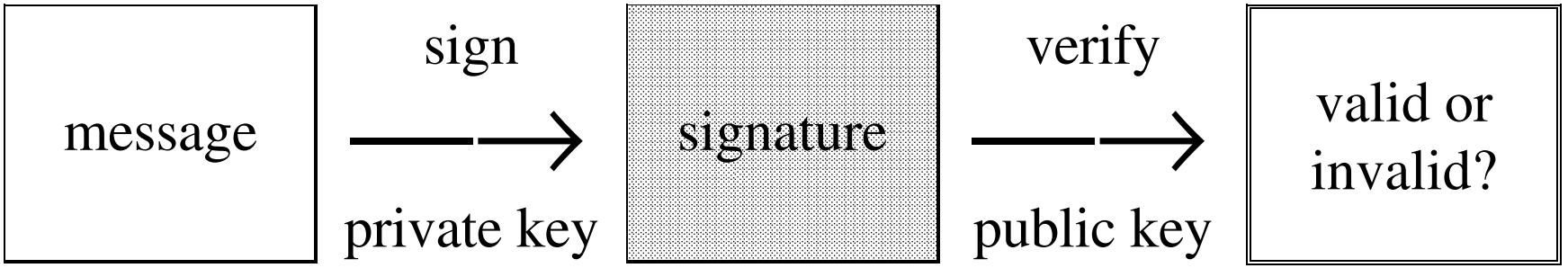
# Introduction

**Digital signatures**

Signature, verification with different keys

For authentication—of message and signer






Users keep one key private, publish other.

# Introduction (cont'd)

**History**

| | |
|---|---|
| 1976 | Diffie, Hellman introduce digital signatures, suggest discrete logarithms as cryptographic problem. |
| 1984 | Elgamal proposes digital signature scheme based on discrete logarithms. |
| 1989 | Schnorr describes efficiency improvement for discrete-logarithm-based schemes. |
| 1991 | NIST announces DSS, a variant of Elgamal with Schnorr improvements. |
| 1992 | NIST revises DSS based on numerous comments. |

# Definition

**System parameters**

$p$: 512-bit prime [revised up to 1024 bits]

$q$: 160-bit prime factor of $p$-1

$g$: proper $q$th root of 1, mod $p$

**Hash function**

$h$: one-way map from message to 160-bit hash

**Keys**

$y$: 512-bit public key

$x$: 160-bit private key

$$y = g^x \bmod p$$

# Definition (cont'd)

**Signature**

$m$: message

Signature = $(r,s)$ where

$$r = (g^k \bmod p) \bmod q$$

$$s = k^{-1}(h(m)+xr) \bmod q$$

and $k$ is 160-bit random integer.

**Verification**

Signature $(r,s)$ for message $m$ is valid if and only if:

$$r = (g^{h(m)t}\, y^{rt} \bmod p) \bmod q,$$

where $t = s^{-1} \bmod q$.

# Definition (cont'd)

**Why it works ...**

If

$$r = (g^k \bmod p) \bmod q,\ \ s = k^{-1}(h(m)+xr) \bmod q,$$
$$y = g^x \bmod p,\ \text{ and } t = s^{-1} \bmod q,$$

then

$$\begin{aligned} g^{h(m)t}\, y^{rt} \bmod p &= g^{h(m)t}\, [g^x]^{rt} \bmod p \\ &= g^{[h(m)+xr]t} \bmod p \\ &= g^{[ks]t} \bmod p \\ &= g^k \bmod p. \end{aligned}$$

So

$$(g^{h(m)t}\, y^{rt} \bmod p) \bmod q = (g^k \bmod p) \bmod q = r.$$

# Definition (cont'd)

**Comparison with other systems**

Elgamal: no $q$

$$r = g^k \bmod p;$$

$$s = k^{-1}(h(m)+xr) \bmod p.$$

Schnorr: $q$, "zero knowledge" ideas

$$r = h(\langle g^k \bmod p, m\rangle);$$

$$s = (k+xr) \bmod q.$$

DSS adds $q$ to Elgamal.

# Performance

**Signature**

$$r = (g^k \bmod p) \bmod q$$
$$s = k^{-1}(h(m)+xr) \bmod q$$

Naive methods: 238.5 mod $p$ multiplications
Sliding three-bit windows: 202
With precomputation (Brickell *et al*, 1991): 52

All but one mod $q$ multiplication is *off line*.
Very good on-line speed, good off-line speed.

# Performance (cont'd)

**Verification**

$$t = s^{-1} \bmod q$$

$$r \stackrel{?}{=} (g^{h(m)t}\, y^{rt} \bmod p) \bmod q$$

Naive methods: 477 mod $p$ multiplications

Simultaneous two-bit windows: 246

With precomputation: 229

Fair speed.

# Performance (cont'd)

**Parameter generation**

$p$: 512-bit prime

$q$: 160-bit prime factor of $p$-1

With trial division by primes ≤ 30, then base-2 pseudoprimality test: $56 \times 624 = 34944$ mod $p$ multiplications

**Key generation**

$$y = g^x \bmod p$$

With precomputation: 52 mod $p$ multiplications.

# Performance (cont'd)

**Comparison with other systems**

In 512-bit modular multiplications, with 512-bit keys:

| | **RSA** | **Elgamal** | **DSS** |
|---|---|---|---|
| *signing *off-line* | n/a | 624 | **52** |
| *on-line* | 159 | **1** | **< 1** |
| verification | **2 to 17** | 689 | 229 |
| parameter generation | n/a | 34944 | 34944 |
| *key generation | 4452 | 624 | 52 |

For DSS, all computations that must be done in private (marked *) are fast—good in smart-card applications.

# Security

## Goals

Given message $m$, find a signature.

Or, better yet,

Given public key $y$, find private key $x = \log_g y$.
This is the *discrete logarithm problem*.

# Security (cont'd)

## $q$-based approaches

Example: Baby-step/giant-step method (Shanks)

1. Tabulate $(u, g^{u\sqrt{q}} \bmod p)$ for all $u$, $0 \le u < \sqrt{q}$. $\sqrt{q}$ time, $\sqrt{q}$ space.
2. For each instance, find $v$, where $0 \le v < \sqrt{q}$, such that $yg^{-v} \bmod p$ is in the table, i.e.,

$$yg^{-v} \bmod p = g^{u\sqrt{q}} \bmod p.$$

Then $x = u\sqrt{q}+v$. $\sqrt{q}$ time.

Other methods find $x$ in $\sqrt{q}$ time, constant space, without a table.

# Security (cont'd)

## *p*-based approaches

Example: Index calculus (Adleman *et al*)

Define $L(p) = \exp((1+\varepsilon)\sqrt{(\log p)\ (\log\log p)})$

1. "Tabulate" $(s, \log_g s)$ for all prime $s$, $2 \le s \le L(p)$. $L(p)$ time, $L(p)$ space.
2. For each instance, find $v$ such that all factors of $yg^{-v} \bmod p$ are in the table, i.e.,

$$yg^{-v} \bmod p = s_1^{e1} \times \cdots \times s_k^{e_k} .$$

Then $x = e_1 \log_g s_1 + \cdots + e_k \log_g s_k + v$. $L(p)$ time.

Improved methods find $v$ in $\sqrt{L(p)}$ time, $\sqrt{L(p)}$ space, with similar time for table.

# Security (cont'd)

## *p*-based approaches (cont'd)

Example: Number field sieve (Gordon, 1991)

Time

$$\exp((2.08+\varepsilon)(\log p)^{1/3} (\log \log p)^{2/3})$$

Asymptotically faster than index calculus, not yet practical.

For special $p$, especially effective.

Note: Attacks based on both $p$ and $q$ are unexplored.

# Security (cont'd)

## Cost in MIPS-years

Based on $L(p)$ as instruction count (Rivest, 1991):

| $\log_2 p$ | $L(p)$ | MIPS years |
|---|---|---|
| 512 | $6.7\times10^{19}$ | $2.1\times10^{6}$ |
| 576 | $1.7\times10^{21}$ | $5.5\times10^{7}$ |
| ... | | |
| 960 | $3.7\times10^{28}$ | $1.2\times10^{15}$ |
| 1024 | $4.4\times10^{29}$ | $1.4\times10^{16}$ |

Security is comparable to RSA's.

$2^{80} \approx 1.2\times10^{24}$ (not directly comparable).

# Security (cont'd)

## Other attacks

*Random number recovery:* If $k$ is known, then $r$ and $x$ can be computed:

$$r = (g^k \bmod p) \bmod q;$$

$$x = r^{-1}(ks\text{-}h(m)) \bmod q.$$

Weak random number generator may reveal $x$.

*Hash function attacks:* Finding messages with the same hash should take time $2^{80}$. May take less time if $h$ is weak.

*Special methods:* Forging signatures is not known to require discrete logarithms, but neither are alternative methods known.

# Trap Doors

## What is a trap door?

Given an algorithm for the forward function, it is computationally infeasible to find a simply computed inverse. Only through knowledge of certain *trap-door information* ... can one easily find the easily computed inverse. (Diffie & Hellman, 1976)

Trap door makes a hard inverse easy.

## Does DSS have a trap door?

Whoever selects *system parameters*, may select *system* trap door.

# Trap Doors (cont'd)

**System trap door**

$p$ of special form (Haber & Lenstra, 1991)

$$p = \sum_{i=0}^{d} p_i m^i$$

where $m$ is an integer and $d$, $p_0$, …, $p_d$ are small

Number field sieve especially effective:

$$\exp((1.00475+\varepsilon)(\log p)^{2/5} (\log \log p)^{3/5})$$

But the smaller the ε, the more obvious the trap door (Gordon, 1992).

# Trap Doors (cont'd)

**Avoiding system trap doors**

1. Choose unique parameters.
2. Trust the one who selects system parameters.

NIST recommends generating $p$ with hash function.

# Conclusions

**DSS: Digital Signature Standard**

- Based on discrete logarithms
- Variant of ElGamal, Schnorr

**Performance**

- Very good on-line signature speed, fair verification speed

**Security**

- Strong to very strong, by current estimates

**Trap doors**

- Possible, but easily avoided