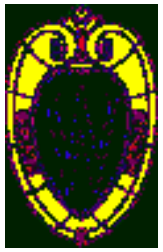

CS/ECE/Math 435
Introduction to Cryptography

Professor Chris DeMarco
Department of Electrical & Computer Engineering
University of Wisconsin-Madison

Spring Semester 2021

Adapting RSA to provide digital signature functionality.

General goals of a digital signature function:

Without prior exchange of a secret key between document author and document reader, we wish to provide the reader with:

1) Confirmation of author identity (preventing another entity from falsely claiming authorship).

2) Confirmation that document retains the exact content it had at signing (preventing modifications to document, post-signature)

---

3) Confirmation of time/date of signing - but this is typically encompassed by (2), provided that part of the document content is a verifiable timestamp.

RSA-based techniques focus on (1) & (2).

Assumption for treatment here:

Document / message to be signed "fits" within one block of the RSA encryption scheme. Recall that RSA block size in bits is set by selected prime number parameter "p".

Subsequent lectures will examine use of hash functions to relax the assumption above, enabling RSA methods to provide digital signature on longer documents / messages.

RSA Digital Signature Protocol

Message author/signer first selects large primes p, q, setting $N = p \cdot q$, and $\varphi = (p-1) \cdot (q-1)$. Next selects integers d, e, such that $\text{mod}(d \cdot e, \varphi) = 1$.

Signer publishes (d, N); p, q, e remain internally secret to signer.

A signed message is pair (x, y), where x is original message, $y = \text{mod}(x^e, N)$.

Reader wishing to verify message tests:

$$\mathrm{mod}(y^d, N) \stackrel{?}{=} x$$

Again, test above really just confirms consistency of x and y with public key (N, d), confirming that $y = \mathrm{mod}(x^e, N)$. But reader must have trust that the identity of the signer really is linked to the public key (N, d).

El-Gamal Digital Signature Scheme

Initialization: • Large prime p, and generator g for $\mathbb{Z}_p^*$ are chosen, and made public.

• An individual user, A, chooses an internally secret $a \in \mathbb{Z}_p^*$, and makes public their $y = \mathrm{mod}(g^a, p)$ { A's public identifier }

• "Document" to be signed is a message $m \in \mathbb{Z}_{p-1}$ (notes: (i) hashing method to accomodate longer messages is our next topic; (ii) $m \in \mathbb{Z}_{p-1}$, not $\mathbb{Z}_p$)

User A's signature on a message/document m is a pair (r, s), $r \in \mathbb{Z}_p$, $s \in \mathbb{Z}_{p-1}$, with r and s satisfying "validity test":

$$\mathrm{mod}(g^m, p) = \mathrm{mod}(y^r \cdot r^s, p)$$

- g: "system-wide" publically known generator
- y: user A's public identifier

- Any user B receiving signed document (i.e., receiving m, r, s) can execute validity test, at a computational cost of order $\log_2(p)$ multiplies, mod p.

- Conversely, an attacker, seeking to forge a signature, must compute "a" from publically known information. Computational cost order $\sqrt{p}$ (i.e. attacker must solve discrete logarithm)

Q: How does user A compute (r, s) satisfying the validity test?

A: A chooses random $k \in \mathbb{Z}_{p-1}^*$.

Observe that by definition of $\mathbb{Z}_{p-1}^*$ it then follows that $\gcd(k, p-1)$ is $= 1$.

And, as per HW II, prob 1(c) (see also Bach notes p. 37-1),

$\gcd(k, p-1) = 1$ and g a generator

$\Rightarrow$ $\boxed{r = \mod(g^k, p)}$ is a generator (for $\mathbb{Z}_p^*$)

A's: choice for r

Then A chooses

$$s = \mathrm{mod}\left(k^{-1}(m - ar),\ p-1\right) \quad (\#)$$

↑ recall $k \in \mathbb{Z}^*_{p-1}$

To confirm that these satisfy validity condition, note that (#) above

$$\Rightarrow \quad \mathrm{mod}(ks + ar,\ p-1) = \underbrace{m}$$

remember that $m \in \mathbb{Z}_{p-1}$; no need for mod operation on rhs

$$\Rightarrow \quad g^m = g^{ar+ks} = (g^a)^r (g^k)^s = y^r \cdot y^s$$

Toy example (CAUTION: Back notes version of this example contains an error!)

$p = 13$, with $g = 2$ as generator for $\mathbb{Z}_{13}^*$. As a generator, $\text{mod}(g^x, 13)$ should be a one-to-one, onto function $\mathbb{Z}_{13}^*$ to $\mathbb{Z}_{13}^*$; check:

$x \in \mathbb{Z}_{13}^*$	1	2	3	4	5	6	7	8	9	10	11	12
$\text{mod}(2^x, 13)$	2	4	8	3	6	12	11	9	5	10	7	1

✓

Recall: p and g are system-wide publically-known parameters.

User A selects internally secret $a = 11$, and publishes A's identifier

$$y = g^a = \mod(2^{11}, 13) = 7.$$

Now, suppose we have document/message $m = 10$.

To sign this document, user A randomly selects a $k \in \underbrace{\mathbb{Z}^*_{p-1}}_{12} = \{1, 5, 7, 11\}$

suppose A selects $k = 5$.

Then A computes

$$r = \mathrm{mod}(g^k, p) = \mathrm{mod}(2^5, 13) = 6$$

$$s = \mathrm{mod}\left[\left(\underbrace{\mathrm{inv}_{\mathbb{Z}_{p-1}}(5)}_{k^{-1}} \cdot \underbrace{(10 - 11 \cdot 6)}_{m - a \cdot r}\right), \underbrace{12}_{p-1 \text{ here!}}\right]$$

$$= \mathrm{mod}\left[\left(5 \cdot (-56)\right), 12\right] = 8$$

NOTE: Bach notes erroneously computes $\mathrm{inv}_{\mathbb{Z}_p}(5) = 8$ on p. 42-2

error in computing inverse in $\mathbb{Z}_p$, instead of $\mathbb{Z}_{p-1}$

So $m = 10$ is accompanied by A's signature of $(r = 6, s = 8)$.

User B receiving signed document performs signature validity test:

$$\underbrace{\text{mod}(g^m, p)} \stackrel{?}{=} \text{mod}(y^r \cdot r^s, p)$$

$$10 = \text{mod}(2^{10}, 13) \stackrel{?}{=} \text{mod}(2^6 \cdot 6^8, 13)$$

$$= \text{mod}(\text{mod}(2^6, 13) \cdot \text{mod}(6^8, 13), 13)$$

$$= \text{mod}(12 \cdot 3, 13) = 10$$

VALIDITY PASSED

Cryptographic Hash Functions

Generally, hash functions take bit strings of arbitrary length "*" to bit strings of specified length "n." Typically assume * > n as the useful case.

Motivation in Digital Signature Context

IF desirable properties of digital signatures can be maintained, we'd like to use previously described algorithms (suitable only for $m \in \mathbb{Z}_{p-1}$, so n must satisfy $2^n \leq p$)

We'll start from list of properties of hash function itself, and then argue that these result in desirable properties for "shortcut" digital signature.

Hash function h must satisfy:

0) $h: \mathbb{Z}_2^* \rightarrow \mathbb{Z}_2^n$

again, denotes arbitrary integer $> n$

Then desirable properties of h:

1) "Forward" computation, given $x \in \mathbb{Z}_2^*$ compute $y = h(x)$, should have "low" computational cost

2) Reverse, <u>pre-image</u> computation should have high computational cost:

given $y \in \mathbb{Z}_2^n$, find <u>an</u> $x \in \mathbb{Z}_2^*$ s.t. $h(x) = y$ [note: because dimension * > dimension n, we can not expect h invertible; two distinct x, x' may map to single y]

3) Collision identification should have high computational cost; i.e. problem of identifying pair $x, x' \in \mathbb{Z}_2^*$, $x \neq x'$, s.t. $h(x') = h(x)$.

4) Second pre-image identification should have high computational cost: Given known x, y pair satisfying $y = h(x)$, compute second $x' \neq x$ s.t. $y = h(x')$.

Hash terminology: x is "message", $h(x)$ is "hash image" or "message digest."

Related observations

i) We'll typically assume $h(\cdot)$ is onto; that every $y \in \mathbb{Z}_2^n$ has at least one $x \in \mathbb{Z}_2^*$ s.t. $y = h(x)$.

ii) And given $x \in \mathbb{Z}_2^*$, sets of form

$$C_x = \{ x' \mid h(x') = h(x) \}$$

partion the set $\mathbb{Z}_2^*$ into equivalence classes; in terminology of hash functions, these C_x sets termed "fibers."

Cryptographic Hash Functions typically implemented by finite-state automata.

In particular, as "stepping stone" towards desired hash function, we first define a transition function

$$\delta : \underbrace{\mathbb{Z}_2^n}_{\text{set consistent with that in which our hash outputs reside; Bach denotes "Q"}} \times \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$$

additional finite set; $m < *$ Bach denotes Σ

With our assumption of $* > m$, partition $X \in \mathbb{Z}_2^*$ into l blocks of length n (as usual, one can "pad out" last block if $*$ not integer multiple of m).

$$X = [x_0, x_1, \ldots x_{l-1}]$$

x_i's $\in \mathbb{Z}_2^m$ will play role of inputs to our finite state machine at each "time" $i = 0, 1, 2 \ldots$;

q_i's $\in \mathbb{Z}_2^n$ are our "states."

At each time i, we update state:

$$\underbrace{q_{i+1}}_{\text{new state}} = \delta(\underbrace{q_i}_{\substack{\text{old state} \\ \text{at time } i}}, \underbrace{x_i}_{\substack{\text{input (part of message } x) \text{ at} \\ \text{time } i}})$$

Evaluation of $y = h(x)$:

1) Partition x;
2) Run finite state machine, initialized at time $i = 0$, to final time that yields $q_\ell = \delta(q_{\ell-1}, x_{\ell-1})$

3) Let $y = q_l \in \mathbb{Z}_2^n$

General Observations

- Computing h requires l forward evaluations of δ; because

$\delta : \mathbb{Z}_2^n \times \mathbb{Z}_2^m \to \mathbb{Z}_2^n$

this is simply a Boolean function.

So we achieve "cheap" computation of "forward" hash function h.

- But computation of pre-image of h requires recovery of all "historic" inputs and states of finite state machine from its final state $y = q_l$. This requires search over space of all historic states $\Rightarrow$ very computationally costly.