Digital Signatures

RSA Digital Signatures

Elgamal and DSA

Digital Signatures

FA 2018 Cryptography Seminar

J. David Taylor

October 22, 2018

Purpose of Digital Signatures

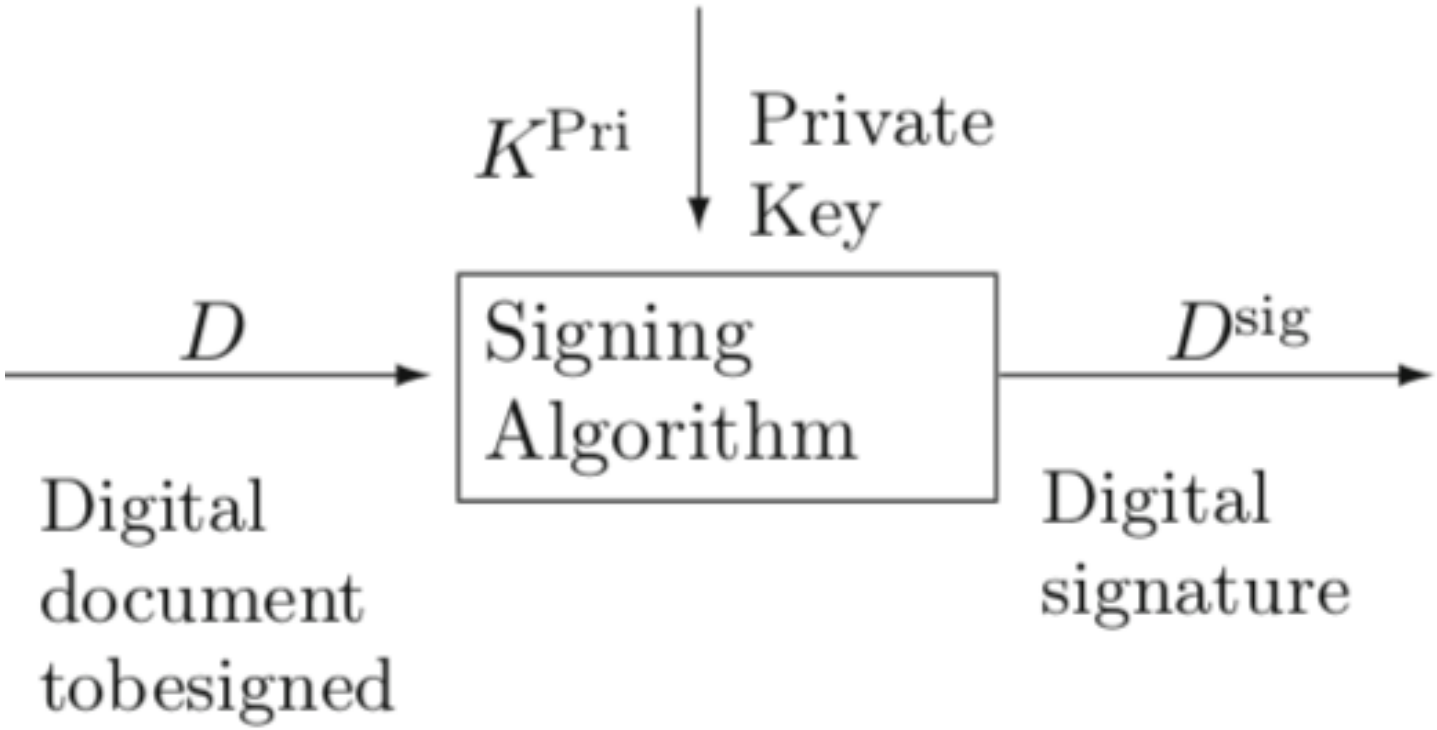
Samantha (the signer) want to "sign" a document. This must satisfy three criteria:

- Samantha's signature must be "attached" to the document, so that it can't just be copied to a new document
- It must be possible for Victor (the verifier) to distinguish Samantha's signature from a forgery
- Checking Samatha's signature shouldn't reveal anything about the document.

Abstract DS Scheme

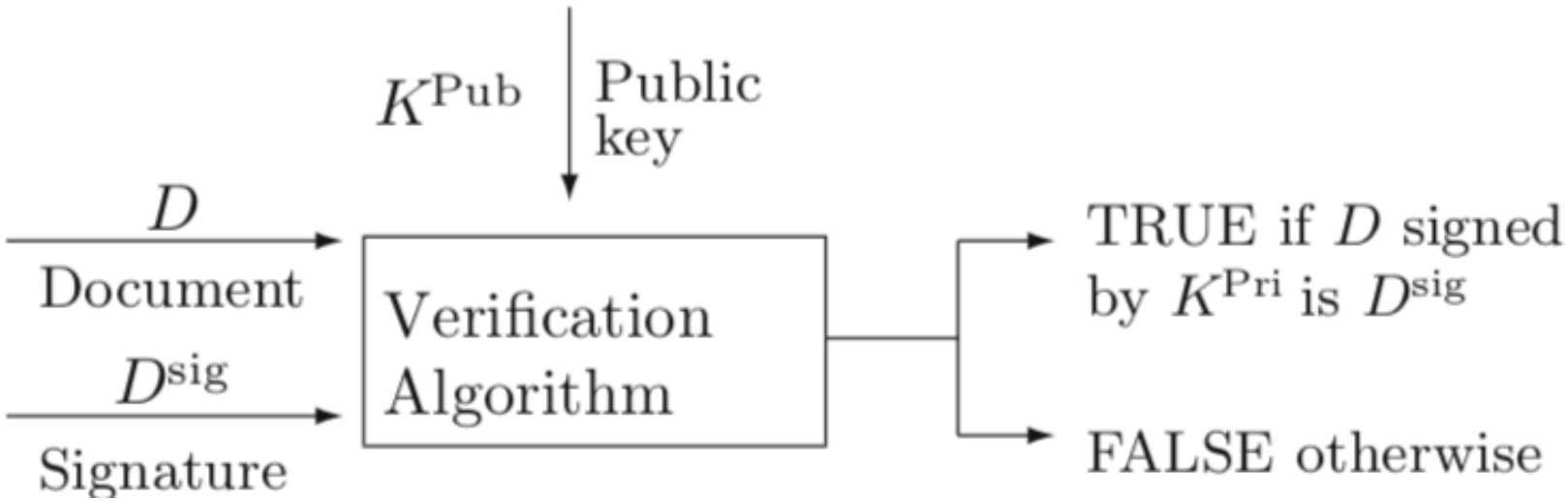
- K^{pri} a private key
- K^{pub} a public key
- *Sign* a signing algorithm that takes a digital document D as input and uses the private key K^{pri} to generate a signature D^{sig}
- *Verify* a verification algorithm that takes the document D, the signature D^{sig}, and the public key K^{pub} as input and returns "True" if D^{sig} is a signature for D and "False" if not.

Signing (Samantha)



k

Verifying (Victor)



Security Conditions

- An attacker cannot determine K^{pri} from K^{pub}, nor can she find another key producing the same signatures as K^{pri}
- Given K^{pub} and a list of signed documents $D_1, D_1^{sig}, D_2, D_2^{sig}, \ldots, D_n, D_n^{sig}$, an attacker cannot determine the signature for a document that is not on the list.

Uses of Digital Signatures

- Verify that software update came from legitimate source
- Sign legal documents

Hashing

Rather than signing the whole document D, it is common to sign a *hash* of the document.

A *Hash function* takes as input an arbitrarily long document D as input and outputs a bit string H. Desired properties:

- The length of H is small
- Computation of $Hash(D)$ should be fast (linear time)
- Inverting $Hash$ should be difficult (exponential time)
- Collision Resistance: It should be unlikely that $Hash(D_1) = Hash(D_2)$ for a pair of distinct documents D_1, D_2.

Secure Hashing Algorithm

Break document D (with extra bits appended) into 512-bit chunks.
Start with five specific initial values $h_0, \ldots, h_4$.
LOOP over the 512-bit chunks.
    Break a 512 bit chunk into sixteen 32-bit words.
    Create a total of eighty 32-bit words $w_0, \ldots, w_{79}$ by
        rotating the initial words.
    LOOP $i = 0, 1, 2, \ldots, 79$
        Set $a = h_0$, $b = h_1$, $c = h_2$, $d = h_3$, $e = h_4$.
        Compute f using XOR and AND operations on a, b, c, d, e.
        Mix a, b, c, d, e, by rotating some of their bits, permuting
            them, and add f and w_i to a.
    END i LOOP
    Set $h_0 = h_0 + a$, $h_1 = h_1 + b$, $\ldots$, $h_4 = h_4 + e$.
END LOOP over chunks
Output $h_0 \parallel h_1 \parallel h_2 \parallel h_3 \parallel h_4$.

The SHA-1 Hash Algorithm

SHA Variants (Wikipedia)

Comparison of SHA functions view · talk · edit

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Rounds	Operations	Security (in bits) against collision attacks	Capacity against length extension attacks	Performance on Skylake (median cpb)[1] long messages	Performance on Skylake (median cpb)[1] 8 bytes	First published
MD5 (as reference)		128	128 (4 × 32)	512	64	And, Xor, Rot, Add (mod 2^{32}), Or	≤18 (collisions found)[2]	0	4.99	55.00	1992
SHA-0		160	160 (5 × 32)	512	80	And, Xor, Rot, Add (mod 2^{32}), Or	<34 (collisions found)	0	≈ SHA-1	≈ SHA-1	1993
SHA-1							<63 (collisions found[3])		3.47	52.00	1995
SHA-2	*SHA-224*	224	256 (8 × 32)	512	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	112	32	7.62	84.50	2004
	SHA-256	256					128	0	7.63	85.25	2001
	SHA-384	384	512 (8 × 64)	1024	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	192	128 (≤ 384)	5.12	135.75	2001
	SHA-512	512					256	0	5.06	135.50	
	SHA-512/224	224					112	288	≈ SHA-384	≈ SHA-384	2012
	SHA-512/256	256					128	256			
SHA-3	*SHA3-224*	224	1600 (5 × 5 × 64)	1152	24[4]	And, Xor, Rot, Not	112	448	8.12	154.25	2015
	SHA3-256	256		1088			128	512	8.59	155.50	
	SHA3-384	384		832			192	768	11.06	164.00	
	SHA3-512	512		576			256	1024	15.88	164.00	
	SHAKE128	*d* (arbitrary)		1344			min(*d*/2, 128)	256	7.08	155.25	
	SHAKE256	*d* (arbitrary)		1088			min(*d*/2, 256)	512	8.59	155.50	

RSA Digital Signatures

Samantha chooses primes p, q and an integer e so that

$$\gcd(e, (p-1)(q-1)) = 1$$

and publishes $N = pq$ and e (the verification exponent).

To sign the document D, Samantha computes and shares

$$S \equiv D^d \mod N,$$

where $de \equiv 1 \mod (p-1)(q-1)$.

Victor verifies Samantha's signature by computing $S^e \mod N$ and checking that it is the same as D.

RSA Digital Signatures

If an attacker wants to forge Samantha's signature:

- They must find an inverse to e mod N, or
- find collisions, or
- find a trick.

Example

- $p = 1223, q = 1987, e = 948047.$
- Samantha publishes $N = pq = 2430101$ and $e = 948047$
- Note $d = 1051235$ (only Samantha knows this)
- To sign $D = 1070777$, she computes

$$S \equiv D^d \equiv 153337 \mod 2430101$$

- Samantha publishes D and S
- Victor verifies that $D \equiv S^e \mod N$

Elgamal Digital Signature

Samantha picks prime number p, a primitive root g, and a signing exponent a.

She computes $A \equiv g^a \mod p$, and publishes A, p, g.

To sign the document $1 < D < p$, she picks $k \in \mathbb{F}_p^* \setminus \{1\}$ at random and computes

$$S_1 \equiv g^k \mod p \text{ and } S_2 \equiv (D - aS_1) \cdot k^{-1} \mod p - 1$$

Her signature is (S_1, S_2).

Elgamal Digital Signature

Victor verifies the signature by checking that

$$g^D \equiv A^{S_1} S_1^{S_2} \mod p$$

Why does this work?

$$A^{S_1} S_1^{S_2} \equiv g^{aS_1 + kS_2} \equiv g^D \mod p$$

DSA

The Digital Signature Algorithm (DSA) is a modification of Elgamal.

It shortens the signature by working in a subgroup of $\mathbb{F}_p^*$ of prime order q.

Setup:

- Samanth chooses two primes p, q such that $p \equiv 1 \mod q$.
- She chooses $g \in \mathbb{F}_p^*$ of exact order q.
- She chooses secret exponenet a, computes $A \equiv g^a \mod p$
- She publishes (A, p, q, g).

DSA

In practice, $2^{1000} < p < 2^{2000}$ and $2^{160} < q < 2^{320}$.

Samantha can take $g \equiv g_1^{(p-1)/2} \mod p$ for some primitive root $g_1 \mod p$.

To sign the document $1 < D < q$, she chooses random $1 < k < q$ and computes

$$S_1 \equiv \left(g^k \mod p\right) \mod q \text{ and } S_2 \equiv (D - aS_1)k^{-1} \mod q$$

Her signature is (S_1, S_2).

DSA

Victor verifies by computing

$$V_1 \equiv DS_2^{-1} \mod q \text{ and } V_2 \equiv S_1 S_2^{-1} \mod q.$$

Then he checks that

$$S_1 \equiv (g^{V_1} A^{V_2} \mod p) \mod q$$

DSA Example

- Samantha picks $p = 48731, q = 443, g = 5260$ (g was computed from the fact that 7 is a primitive root mod 48731)
- Samantha picks $a = 242$
- Samantha publishes $A \equiv 5260^{242} \equiv 3438 \mod 48731$, p, q, g.

DSA Example

To sign $D = 343$, Samantha uses $k = 427$ to compute and share

$$\begin{aligned} S_1 &\equiv \left(5260^{427} \mod 48731\right) \mod 443 \\ &\equiv 2727 \mod 443 \\ &\equiv 59 \mod 443 \end{aligned}$$

and

$$\begin{aligned} S_2 &\equiv (343 + 242 \cdot 59) \cdot 427^{-1} \mod 443 \\ &\equiv 166 \mod 443 \end{aligned}$$

DSA Example

Victor verifies the signature by computing

$$V_1 \equiv 343 \cdot 166^{-1} \equiv 357 \mod 443$$

and

$$V_2 \equiv 59 \cdot 166^{-1} \equiv 414 \mod 443$$

and then checking that

$$\begin{aligned} g^{V_1} A^{V_2} &= (5260^{357} 3438^{414} \mod 48731) \mod 443 \\ &\equiv 2717 \mod 443 \equiv 59 \mod 443 \end{aligned}$$

So the signature checks out.