Digital Signatures

→ Digital Signatures are crucial for public-key cryptography alongside key establishment (Ex. digital certificates secure web browsing, secure software updates, etc) → provide a method to ensure msg is authentic to one user & provide novel functionality

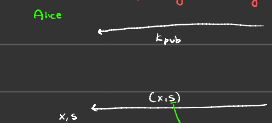
→ There are many security needs besides encryption and key exchange

→ Motivating Example where symmetric cryptography fails to provide desirable security [2]: Alice wants to send Bob a msg, where they share a secret key → use sound symmetric cipher, Oscar won't learn plaintext. However, they can encounter Msg Authentication Code (ensures Bob that msg comes from Alice) based on symmetric cryptography. What if Alice and Bob are against each other? → In many situations where it is important to prove to a neutral third party that one of 2 parties generated a msg.

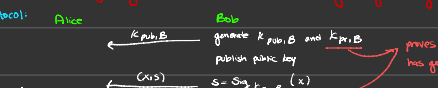
↳ Why symmetric cryptography doesn't work: Alice and Bob have same keys ∴ same capabilities ∴ neutral party cannot distinguish.

↳ Sol'n: Use Public Key Crypt. as the asymmetric setup might enable neutral party to distinguish (Digital Signatures)

→ Basic Idea of Digital Signatures: Person who signs msg does (e.g.) a private key and receiving/using matching public key — principle of DS



→ Basic Digital Signature Protocol:



X must also be encrypted

→ Security Objectives/Services: — CIA Triad

↳ Confidentiality: Info kept secret from all but authorized parties

↳ Integrity: Msgs have not been modified in transit

↳ Message Authentication: Sender of msg is authentic — ensures

↳ Non-repudiation: Sender of msg cannot deny creation of msg

↳ Identification: Establish & verify identity of entity

↳ Access Control: Restrict access to resources to privileged ones

↳ Availability: Ensures access to resources are available

↳ Auditing: Provide evidence about security-relevant activities

↳ Physical Security: Provide protection against physical tampering & attempts

↳ Anonymity: Provide protection against misuse of identity

→ Applications of Digital Signatures:

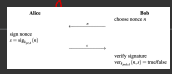
↳ Certificates: Bind identity to public key

↳ Secure Boot and Firmware Updates: Ensure only well-functioning & tested software from provider runs on on a particular computer & cannot change w/o authorization → does this during bootup or updates

↳ Pro: Only public key needs to be installed → requires Unified Extensible Firmware Interface

↳ Proof-of-Knowledge Protocol: If receiver needs to prove authenticity, convince they are of possession of certain resources

Ex.



↳ Pro: Sensitive k_{pr} resides securely inside smart card & verification key doesn't need special protection

→ RSA Signature Scheme: Based on RSA: Security relies on integer factorization problem

↳ role of public & private keys are swapped

→ Signature s is as long as n ($\lceil \log_2 n \rceil$) → n is at least 2048 bits

→ Square-and-Multiply must be used and Acceleration Techniques

→ $n = pq$ are sufficiently large prime p & q

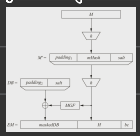
→ Ensure authenticity of (n,e) by verifying party that is associated w/ private key

→ Solution to Existential Forgery: Allow certain msg formats → padding

→ Probabilistic Signature Scheme (RSA-PSS): Combines signing + padding (H = msg. Signed by hashed version)

Encoding for EMSA PSS Scheme: EM (Encoded Msg) is of length $\lceil \frac{(n-1)}{8} \rceil$ bytes s.t. Bit length of EM $\leq |n|-1$ bits

1. generate random Value Salt
2. $H' = Hash(0x00\,00\,00\,00\,00\,00\,00\,00 \,||\, H(M) \,||\, salt)$
3. $H(M)$
4. Concat $padding_2$ + salt = DB
5. Apply Mask Generation f^{as} to $H(M')$ → dbMask
6. masked DB = dbMask ⊕ DB
7. EM = masked DB || H || bc



$S = Sig_{k_{pr}}(M) = EM^d \bmod n$

return $(x, s, salt)$

RSA keys: $k_{pub} = (n, e)$, $k_{pr} = (d)$ — Basic but padding required in practice

Alice		Bob
	$\leftarrow (n,e)$	$k_{pub} = (n,e)$ $k_{pr} = (d)$
Verify	$\leftarrow (x,s)$	$s = Sig_{k_{pr}}(x) \equiv x^d \bmod n$
$ver_{k_{pub}}(x,s)$		
$x' \equiv s^e \bmod n$		
if $x' \equiv x \bmod n \Rightarrow$ valid		authentication & integrity

Proof: Shows integrity holds.

$s^e \equiv (x^d)^e \equiv x^{de} \equiv x \bmod n$

$de \equiv 1 \bmod \beta(n) \Rightarrow x^{de} = x$

→ Existential Forgery: Prevented by RSA signature scheme → adversary allows valid signature for x

Alice	Oscar	Bob
		$k_{pr} = d$
	$\leftarrow (n,e)$	$k_{pub} = (n,e)$
$s^e \equiv x' \bmod n$	$se \in \mathbb{Z}_n$	
$\Rightarrow x' = x \Rightarrow$ valid	$\leftarrow (x,s)$	$x \equiv s^e \bmod n$
		Impersonates Bob

If attacker uses a correct k_{Pub}, can sign msgs. → Use certificates

→ Elgamal Digital Signature Scheme: Consists of a key generation & execution phases

Key Generation:

Choose large prime p → Choose generator α of $\mathbb{Z}_p^*$ or of subgroup of $\mathbb{Z}_p^*$

$\beta \equiv \alpha^d \bmod p$ ← Choose $d \in \{2, 3, \ldots, p-2\}$

$k_{pub} = (p, \alpha, \beta)$ and $k_{pr} = d$

Using k_{pr} and k_{pub},

$Sig_{k_{pr}}(x, k_E) = (r, s)$

Elgamal Signature Generation.

1. Choose ephemeral key $k_E \in \{2, \ldots, p-2\}$ s.t. $gcd(k_E, p-1) = 1$
2. $r \equiv \alpha^{k_E} \bmod p$
   $s \equiv (x - d \cdot r) k_E^{-1} \bmod (p-1)$

On verification side:

1. $t \equiv \beta^r \cdot r^s \bmod p$
2. $t \equiv \alpha^x \bmod p \Rightarrow$ valid; invalid o.w.

Correctness Proof: We show that we get a valid signature if verifier uses correct public key & correct msg & correct (r,s)

$\beta^r \cdot r^s \equiv (\alpha^d)^r (\alpha^{k_E})^s \bmod p$

$\equiv \alpha^{dr + k_E s} \bmod p$

$\equiv \alpha^x \bmod p$

$\Rightarrow x \equiv dr + k_E s \bmod (p-1)$

$\Rightarrow s \equiv (x - dr) k_E^{-1} \bmod (p-1)$

Identical to Elgamal Encryption

→ Security on Elgamal DS Scheme relies on Discrete Logarithm Problem

→ ∴ p should have at least 2048 bits to ensure it can't be computed.

Private Key should be generated by TRNG

Public Key requires Square-and-Multiply

Signature Pair = (r, s) — have roughly same bit length as p ∴ $(x, (r,s))$ is about 3x the length of x

→ Security that depends on verifier using correct public key

→ Security certainty relies on DLP → if Oscar can compute discrete logs, he can compute d from β and k_E from r

↳ ∴ p must be ≥ 2048 bits long

↳ Subgroup attacks must also not be possible — α generator generate subgroups of prime order → all elements in are generators ∴ no subgroups exist

↳ can sign messages on behalf of owner

Inverse of k_E via EEA & speed up using Precomputation — needs SAM to do Modular exponentiation

Signer generates ephemeral key k_E & r in advance → store both

→ If signer reuses k_E, Oscar can do Computation: Oscar sees 2 pairs of form $(x_1, (r,s_1))$ & $(x_2, (r,s_2))$. If x_1 & x_2 have same $k_E \Rightarrow r_1 = r_2 \equiv \alpha^{k_E} \bmod p$

$k_E \equiv \frac{x_1 - x_2}{s_1 - s_2} \bmod (p-1)$ ← $s_1 \equiv (x_1 - dr) k_E^{-1} \bmod (p-1)$, $s_2 \equiv (x_2 - dr) k_E^{-1} \bmod (p-1)$ — Sys. of linear equations

$d \equiv \frac{x_1 - s_1 k_E}{r} \bmod (p-1)$ if $gcd(s_1 - s_2, p-1) \neq 1$, many sol'n & Oscar chooses correct one.

→ Oscar could also generate valid signature for x to impersonate Bob → Attack is not possible if x is hashed $(s = (H(x) + dr) k_E^{-1} \bmod (p-1))$

Alice | Oscar | Bob

$k_{pr} = d$ $k_{pub} = (p, \alpha, \beta)$ → (p, α, β)

1. Selects j s.t. $\gcd(j, p-1) = 1$
2. $r \equiv \alpha^j \bmod p$
   $s \equiv -r j^{-1} \bmod (p-1)$
3. $x \equiv s\, i \bmod (p-1)$

← (p, α, β)

Verification
$t \equiv \beta^r \cdot r^s \bmod p$
$\equiv \alpha^x \bmod p$

→ Proof of the fact that modified Signatures are invalid:

WTS that (r,s) satisfies $V \equiv r \bmod q$

$s \equiv (SHA(x) + dr)\, k_E^{-1} \bmod q$

$\Rightarrow k_E \equiv s^{-1} SHA(x) + ds^{-1} r \bmod q$

$k_E \equiv U_1 + dU_2 \bmod q$

$\alpha^{k_E} \bmod p \equiv \alpha^{U_1 + dU_2} \bmod p$

$\equiv \alpha^{U_1} \beta^{U_2} \bmod p$

$(\alpha^{k_E} \bmod p) \bmod q \equiv (\alpha^{U_1} \beta^{U_2} \bmod p) \bmod q$

$r \equiv V \bmod q$

→ Most demanding component is Key Generation Phase → only in set-up

Challenge: Find $\mathbb{Z}_p^*$ with bit length 2048 and subgroup of 2^{224} range → fulfilled if $p-1$ has prime factor q

General Approach: Find 224-bit q ⇒ construct p from it & uses Miller-Rabin primality test

Prime Generation for DSA:

Output: 2 primes (p, q) s.t. $p-1$ is multiple of q

Alg:

1. Find q using Miller-Rabin
2. For $i = 1$ to 4096:
   generate M with $2^{2047} < M < 2^{2048}$
   $M_r \equiv M \bmod 2q$
   $p-1 \equiv M - M_r$
   If p is prime: return (p, q)

Go TO STEP 1

→ Elliptic Curves require primes of bit lengths 256 - 512 bits → security equiv to RSA & DL schemes of 2048+

→ ECDSA Signatures consist of pairs (r,s) → each value has same bit length

ECDSA Signature Generation:

1. Choose random $0 < k_E < q$
2. $R = k_E A = (x_R, y_R)$
3. $r = x_R$
4. $s \equiv (h(x) + dr) k_E^{-1} \bmod q$

hash f^n output (length at least $\lg q$)

ECDSA Signature Verification:

1. Compute aux $w \equiv s^{-1} \bmod q$
2. Compute aux $U_1 \equiv w \cdot h(x) \bmod q$
3. Compute aux $U_2 \equiv w \cdot r \bmod q$
4. Compute $P = U_1 A + U_2 B$
5. $\mathrm{Ver}_{k_{pub}}(x, (r,s)) \Rightarrow x_P \equiv$ { $r \bmod q \Rightarrow$ valid; $o.w \Rightarrow$ not valid } (x_P = x-coord of point P)

→ Computations in each stage of ECDSA:

1. Key Generation: EC with prime p of $|p| \geq 256$
2. Signing: Compute point R via point mult., $s \Rightarrow$ invert $k_E \Rightarrow$ EEA
   we need to hash x & mod q
3. Verification: Compute $w, U_1, U_2 \Rightarrow$ main is $P = U_1 A + U_2 B$ (2 point mult)

→ Digital Signature Algorithm is more popular, and is better than ElGamal since signature shorter than modulus

↳ Main idea: 2 cyclic groups involved. $\mathbb{Z}_p^* \Rightarrow \mathrm{ord}(\mathbb{Z}_p^*) = 2048$ bits
$\mathbb{Z}_q^* \Rightarrow \mathrm{ord}(\mathbb{Z}_q^*) = 224$

Key Generation for DSA:

1. Choose prime p with $2^{2047} < p < 2^{2048}$
2. Find prime divisor q of $p-1$ with $2^{223} < q < 2^{224}$
3. Find α with $\mathrm{ord}(\alpha) = q$ (α generates subgroup of α elem)
4. Find random $d \in \mathbb{Z}^+$ with $0 < d < q$
5. Compute $\beta \equiv \alpha^d \bmod p$

$k_{pub} = (p, q, \alpha, \beta)$

$k_{pr} = d$

DSA consists of (r,s) → each being 224 bits long

1. Choose $k_E \in \mathbb{Z}^+$ with $0 < k_E < q$
2. $r \equiv (\alpha^{k_E} \bmod p) \bmod q$
3. $s \equiv (SHA(x) + dr) k_E^{-1} \bmod q$ → SHA-2

DSA Signature Verification:

1. Compute $w \equiv s^{-1} \bmod q$
2. $U_1 \equiv w \cdot SHA(x) \bmod q$
3. $U_2 \equiv w \cdot r \bmod q$
4. $V \equiv (\alpha^{U_1} \cdot \beta^{U_2} \bmod p) \bmod q$
5. $\mathrm{Ver}_{k_{pub}}(x, (r,s)) = V \equiv$ { $r \bmod q \Rightarrow$ valid; $o.w \Rightarrow$ not valid }

Now that we have p & q we need to protect against 2 DLP Attacks → If Oscar wants to attack, could compute $d \equiv \log_\alpha \beta \bmod p$

Most Powerful Attack: Index-Calculus Attack → Solve: p must be ≥ 2048 bits

Could also exploit α generating small subgroup of order q → not applicable if Oscar wants to exploit subgroup property

Perform generic DLP attack → Hash Function must also maintain Security of DLP

→ DSA becomes vulnerable if ephemeral key is reused

→ steps in ECDSA similar to DSA → DLP constructed using Elliptic Curves → ∴ operations are different

→ ECDSA is defined for Elliptic Curves over $\mathbb{Z}_p$ prime fields and $GF(2^m)$ Galois Fields

Key Generation for ECDSA:

1. Use E with modulus p, coeff. a & b, point A generating cyclic group of prime order q
2. Compute random $0 < d < q$
3. Compute $B = dA$

$k_{pub} = (p, a, b, q, A, B)$

$k_{pr} = (d)$

Proof: WTS (r,s) satisfies ECDSA verification condition: $r \equiv x_P \bmod q$

$s \equiv (h(x) + dr) k_E^{-1} \bmod q$

$k_E \equiv s^{-1} h(x) + ds^{-1} r \bmod q$

$k_E \equiv U_1 + dU_2 \bmod q$

$k_E A = (U_1 + dU_2 \bmod q) A$

$k_E A = U_1 A + dU_2 A$

$k_E A = U_1 A + U_2 B$

→ Security of ECDSA: If Oscar attempts to solve ECDLP to get private key d and for $k_E \Rightarrow$ successful

Hash f^n must be good

q	Hash output (min)	Security levels
192	192	96
224	224	112
256	256	128
384	384	192
512	512	256