Asymmetric Cryptography Standards

Alexandre Duc

Textbook RSA (Plain RSA)

- Generate two random secret prime numbers p and q. Compute $n = pq$.
- Choose a small number e coprime with $\varphi(n) = (p-1)(q-1)$.
- Compute $d = e^{-1} \bmod \varphi(n)$, and erase p, q and $\varphi(n)$.
- **Public key** : (n, e). **Private key** : (n, d).
- **Encryption** : $c = m^e \bmod n$.
- **Decryption** : $m = c^d \bmod n$.

Small Exponent

We decide to encrypt an AES 128-bit key with textbook RSA. The AES key will be then used to encrypt data. To make things secure, we decide to select a 2048-bit RSA modulus and $e = 3$.
What attack can you do ?

Small Exponent

Solution

Since the exponent e and the plaintext m are small, m^e (over the reals) is still smaller than the modulus. Hence, the modulus has no effect here. It is, thus, possible to recover easily m from the ciphertext c by computing $\sqrt[3]{c}$. There are simple numerical algorithms doing so (e.g. Newton's method).

Small Exponent and Broadcasting

Suppose that we use textbook RSA with $e = 3$ to send the same message to three different participants.
All three participants have a different RSA modulus, the RSA moduli are 2048-bit long and the message that is sent is also 2048 bit long.
What attack can you perform ?

Small Exponent and Broadcasting

Solution

We will use the Chinese Reminder Theorem (CRT) to increase the space of the message. We have at our disposal three ciphertexts $c_1 = m^e \bmod n_1$, $c_2 = m^e \bmod n_2$, and $c_3 = m^e \bmod n_3$. Using CRT, we obtain a new ciphertext $c = m^e \bmod n_1 n_2 n_3$. The exponent is now much bigger and we can perform the same attack as in the previous question.

Small Exponents

WARNING

Always avoid small exponent, even if the message is formatted. Coppersmith's attack allows to decrypt when e is small.

A typical (good) choice is $e = 65537$

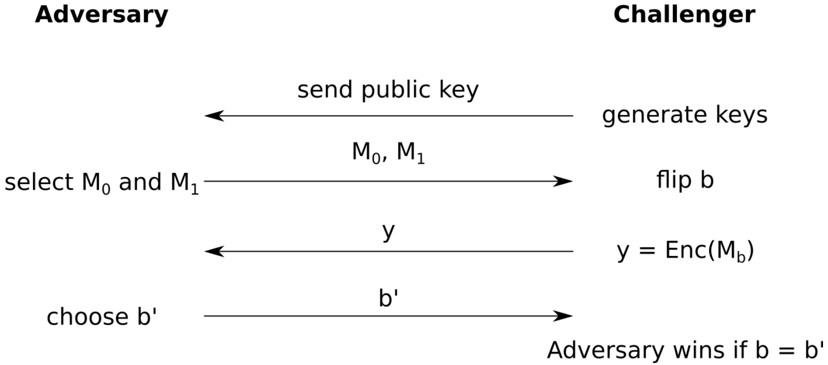
Small Private Keys

Warning

Small private keys are also bad.
Wiener key recovery attack : for $d < \sqrt[4]{N}$.

Don't fix a private key. The inverse of 65537 is extremely likely to **not** be too small.

Indistinguishability under Chosen-Plaintext Attacks (IND-CPA)



- A cryptosystem is said to be **indistinguishable under chosen-plaintext attack (IND-CPA)** if every *efficient* adversary has only a negligible advantage over random guessing.

IND-CPA Security (formal)

IND-CPA Security

A cryptosystem is IND-CPA secure if $\Pr[\text{win IND-CPA game}] - \frac{1}{2}$ is **negligible** for every PPT (probabilistic polynomial time) adversary.

Textbook RSA : IND-CPA secure ?

Question

Is Textbook RSA IND-CPA secure ?

Textbook RSA : IND-CPA secure ?

Solution

No, it is not. Since we possess the public key, we can try to encrypt both M_0 and M_1 and check which one matches y.

RSA PKCS v1.5

- Older version of the standard RSA encryption and signature padding method.
- Standardized in PKCS#1 v1.5 and RFC 2313.
- Format : `EB = 00 || BT || PS || 00 || D`, where :
  - `BT` is the block type and can be equal to `00`, `01` or `02` ;
  - `PS` is a string of `00`'s (if `BT==00`), or of `FF`'s (if `BT==01`) or of non-zero pseudo-random bytes (if `BT==02`) of at least 8 bytes ;
  - `D` are the data bytes to be encrypted.
- None of the versions of PKCS#1 v1.5 are IND-CPA secure !
- **RSA PKCS#1 v1.5 should be avoided in all new applications !**

Bleichenbacher Attack and Chosen-Ciphertext Security

- Daniel Bleichenbacher, a Swiss cryptographer, has exhibited the first **adaptive chosen-ciphertext** attack against RSA PKCS v1.5 in 1998.
- Chosen-ciphertext security has transformed itself from a theoretical to a practical concern.
- 2017 : ROBOT attack. Attack on SSL/TLS using Bleichenbacher's attack.
- Typical case of **oracle attack**.

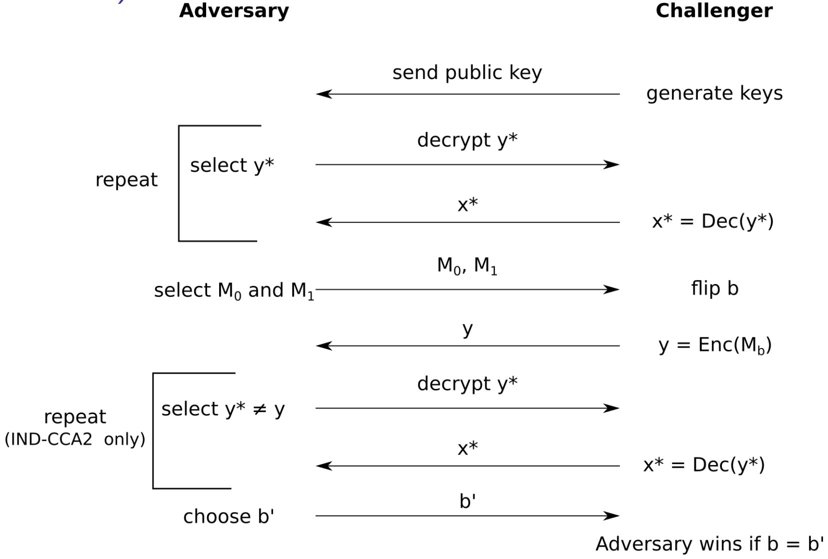
Bleichenbacher Attack : Details

- Let's suppose that `BT = 02` (most common).
- Idea : multiply unknown ciphertext by s^e and use oracle to check if it is valid.
- A valid ciphertext starts with `0x0002`. $\rightarrow$ ms starts with `0x0002`.
- We have $0002 \cdots \leq ms < 0003 \ldots$.
- With a binary search technique and thousands of queries : can **decrypt ciphertext**.

Warning

Typical **implementation problem** : padding oracle attack.

Indistinguishability under Chosen-Ciphertext Attacks (IND-CCA)



IND-CCA Security (formal)

- Two versions : non-adaptive (IND-CCA) and adaptive (IND-CCA2).

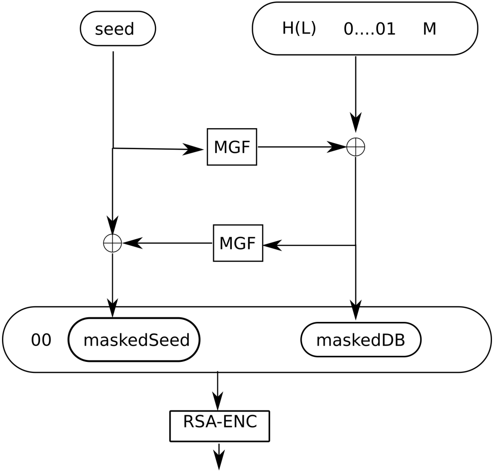
IND-CCA Security

A cryptosystem is IND-CCA secure if Pr[win IND-CCA game] $-\frac{1}{2}$ is **negligible** for every PPT (probabilistic polynomial time) adversary.

RSA-OAEP

- Improved version proposed by Bellare and Rogaway in 1994.
- Padding shown to be IND-CCA2 secure when used with the RSA permutation
- Design goals :
  - Add randomness
  - Prevent partial decryption of ciphertexts : an adversary cannot recover **any part of the plaintext** without inverting the underlying trapdoor one-way function.
- Standardized in PKCS#1 v2.1 and v2.2 as well as in RFC 3447

RSA-OAEP



Manger's Attack on RSA-OAEP

- RSA-OAEP is IND-CCA secure against **black-box** adversaries.
- Manger's attack : Similar to Bleichenbacher's in the idea.
- Two error messages should be **identical**.
- Problem : timings, typos, ...

keylength.com

Method	Date	Symmetric	Factoring Modulus	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash
[1] Lenstra / Verheul	2022	87	1995 1568	154	1995	164	174
[2] Lenstra Updated	2022	83	1446 1660	166	1446	166	166
[3] ECRYPT	2018 - 2028	128	3072	256	3072	256	256
[4] NIST	2019 - 2030	112	2048	224	2048	224	224
[5] ANSSI	2021 - 2030	128	2048	200	2048	256	256
[6] NSA	-	256	3072	-	-	384	384
[7] RFC3766	-	-	-	-	-	-	-
[8] BSI	2020 - 2022	128	2000	250	2000	250	256

All key sizes are provided in bits. These are the minimal sizes for security.

- There exist many tables.
- It is up to you to choose which ones you trust.

ECRYPT

The goal of ECRYPT-CSA (Coordination & Support Action) is to strengthen European excellence in the area of cryptology. This report [3] on cryptographic algorithms, schemes, keysizes and protocols is a direct descendent of the reports produced by the ECRYPT I and II projects (2004-2012), and the ENISA reports (2013-2014). It provides rather conservative guiding principles, based on current state-of-the-art research, addressing construction of new systems with a long life cycle. This report is aimed to be a reference in the area, focusing on commercial online services that collect, store and process the data.

Protection	Symmetric	Factoring Modulus	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash
Legacy standard level *Should not be used in new systems*	80	1024	160	1024	160	160
Near term protection *Security for at least ten years (2018-2028)*	128	3072	256	3072	256	256
Long-term protection *Security for thirty to fifty years (2018-2068)*	256	15360	512	15360	512	512

All key sizes are provided in bits. These are the minimal sizes for security.
Click on a value to compare it with other methods.

Recommended algorithms:
Block Ciphers: For near term use, AES-128 and for long term use, AES-256.
Hash Functions: For near term use, SHA-256 and for long term use, SHA-512 and SHA-3 with a 512-bit result.
Public Key Primitive: For near term use, 256-bit elliptic curves, and for long term use 512-bit elliptic curves.

Future algorithms (expected to remain secure in 10-50 year lifetime):
Block Ciphers: AES, Camellia, Serpent
Hash Functions: SHA2 (256, 384, 512, 512/256), SHA3 (256, 384, 512, SHAKE128, SHAKE256), Whirlpool-512, BLAKE (256, 584, 512)
Stream Ciphers: HC-128, Salsa20/20, ChaCha, SNOW 2.0, SNOW 3G, SOSEMANUK, Grain 128a

Elliptic Curve Choice

- Which elliptic curve should I choose ?
- Many different curves : some have multiple different names.
- Three different categories : **Weierstrass, Montgomery, Edwards**.
- Weierstrass curves (seen in class) : $y^2 = x^3 + ax + b$
- Montgomery curves : $By^2 = x^3 + Ax^2 + x$
- Twisted Edwards curves : $ax^2 + y^2 = 1 + dx^2y^2$. **No** point at infinity. The point $(0, 1)$ is the neutral element.

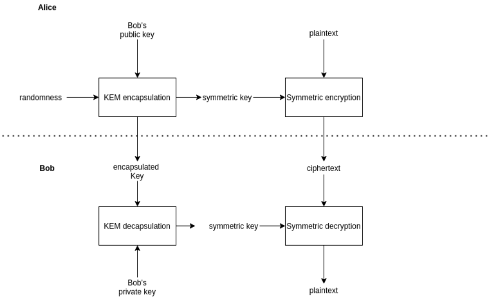
Which Type ?

- Twisted Edwards curves can be mapped to Montgomery form.
- Montgomery curves can be mapped to Weierstrass form.
- **Not** all Weierstrass curves are Montgomery (or Edwards).
- The mappings are described here : `https://tools.ietf.org/id/draft-struik-lwip-curve-representations-00.html`
- Classical **double-and-add** algorithm is vulnerable to side-channel attacks.
- Solution : **Montgomery ladder** : very efficient for Montgomery (and Edwards) curves.
- Twisted Edwards are needed for **EdDSA**.

Recommendation

Curve	Legacy	Future	Remark
		Weierstrass	
W-25519, W-448	✓	✓	From Montgomery curves
P-256, P-384, P512	✓	✓	Hard to implement. Used almost everywhere
P-192, P-224	✓	x	too small
		Montgomery	
Curve25519	✓	✓	
Curve448	✓	✓	
		Twisted Edwards	
Edwards25519	✓	✓	
Edwards448, E448	✓	✓	
Binary fields	x	x	**Broken**. Favor underlying prime fields

Hybrid Encryption



- **Hybrid encryption** combines an asymmetric encryption algorithm with a symmetric one.
- Two parts : a **Key encapsulation mechanism (KEM)** and a **Data encapsulation mechanism (DEM)**.
- The KEM is asymmetric and encrypts a symmetric key.
- The DEM is a symmetric encryption algorithm.
- **Typical usecase** : encrypted emails.

ECIES

- The **Elliptic Curve Integrated Encryption Scheme (ECIES)** is an integrated encryption scheme that uses the following functions : a key agreement protocol, a key derivation function, a hash function, an symmetric encryption scheme and a MAC.
- Standardized in ANSI X9.63, IEEE 1363a, ISO 18033-2 and SECG SEC 1.
- All the standardized versions show minor differences...

ECIES Setup

- ECIES allows **hybrid encryption** : uses public key crypto to exchange a symmetric key and use symmetric crypto later to send data.
- Public parameters : an elliptic curve E, a point G on E of order n (prime), KDF, symmetric encryption scheme Enc, MAC.
- **Secret key :** $k \in \mathbb{Z}_n$. **Public key :** $K = kG$.

ECIES Encryption(m)

1. Draw $r \in \mathbb{Z}_n^*$ uniformly at random.
2. Let $R = rG$.
3. $(k_E \| k_M) = \mathrm{KDF}(rK)$.
4. $c = \mathrm{Enc}_{k_E}(m)$
5. $\tau = \mathrm{MAC}_{k_M}(c)$.
6. Ciphertext is $R \| c \| \tau$.

ECIES Decryption($R \| c \| \tau$)

Question

How do you decrypt ?

ECIES Decryption($R\|c\|\tau$)

Solution

1. $(k_E\|k_M) = \mathrm{KDF}(kR)$.
2. Verify that $\tau = \mathrm{MAC}_{k_M}(c)$.
3. If correct : $m = \mathrm{Dec}_{k_E}(c)$

RSA-KEM

- **Hybrid encryption** based on RSA.
- See RFC 5990.
- Algorithm :
  1. Draw a random number $z \in \mathbb{Z}_n$.
  2. Let $u = z^e \bmod n$
  3. Use $(k_E \| k_M) = \mathrm{KDF}(z)$ for the data.
  4. $c = \mathrm{Enc}_{k_E}(m)$
  5. $\tau = \mathrm{MAC}_{k_M}(c)$
  6. Ciphertext is (u, c, τ).

Encryption Summary

Warning

Warning with the **implementation** of RSA-OAEP.

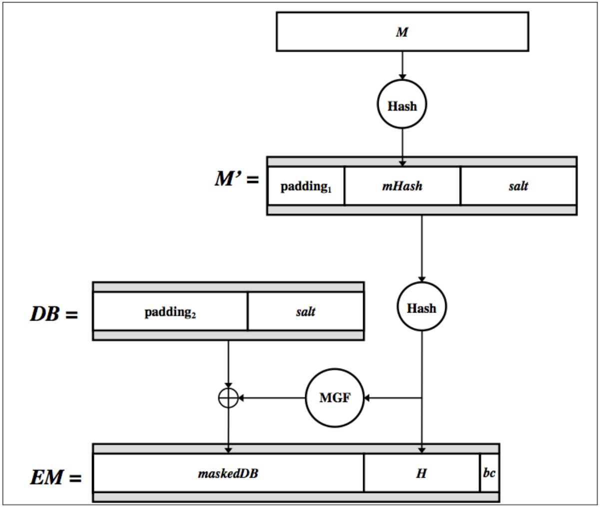
Primitive	Legacy	Future
RSA-OAEP	✓	✓
RSA-KEM	✓	✓
ECIES	✓	✓
RSA-PKCS#1 v1.5	x	x

source of recommendations : https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf

RSA-PSS

- RSA-PSS uses the Probabilistic Signature Scheme (PSS) proposed by Bellare and Rogaway.
- Standardized in PKCS#1 v2.1 and v2.2 as well as in RFC 8017.
- Relies on randomization and hash functions.

RSA-PSS


M
Hash
M' =
padding1
mHash
salt
DB =
padding2
salt
Hash
MGF
EM =
maskedDB
H
bc

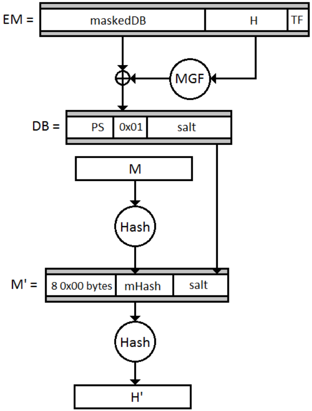
RSA-PSS Verification

Question

How do you verify the signature ?

RSA-PSS Verification

Solution



DSA

- DSA stands for **Digital Signature Algorithm**
- DSA is a variant of El-Gamal signatures and its security relies on the discrete logarithm problem in $\mathbb{Z}_p$.
- DSA is standardized in NIST FIPS 186-4 (known as the "Digital Signature Standard (DSS)") and withdrawn in FIPS 186-5.

DSA : Key Generation

- We work in $\mathbb{Z}_p^*$ with an element $g \in \mathbb{Z}_p^*$ of prime order q.
- Private Key : $a \in \mathbb{Z}_q$, public key : $A = g^a \bmod p$.

DSA : Signature

- We use a cryptographically secure hash function $h : \{0,1\}^* \longrightarrow \{1,\dots,q-1\}$.
- We generate a uniformly random number $k \in \{1,\dots,q-1\}$.
- To sign a message m, we compute $r = (g^k \bmod p) \bmod q$ and $s = k^{-1}(h(m) + ar) \bmod q$.
- The signature of m is (r, s) if $r \neq 0$ and $s \neq 0$. Otherwise, restart with a fresh k.

DSA : Verification

- We first **verify** that $0 < r < q$ and $0 < s < q$.
- To verify the signature (r, s) attached to a message m, we verify that $r = \left(g^{h(m)s^{-1}} A^{rs^{-1}} \bmod p\right) \bmod q$.

ECDSA

- ECDSA is a variant of DSA working on elliptic curves.
- It is standardized in NIST FIPS 186-4, X9.62 and SEC2.
- Used in TLS 1.x and SSH
- More and more seen. Will replace DSA.

ECDSA parameters

- We adapt the DSA signature algorithm to elliptic curves (obtaining the ECDSA algorithm).
    1. Choose a cryptographically secure elliptic curve and a point G of order n.
    2. **Private Key** : $a \in \{1, \ldots, n-1\}$.
    3. **Public Key** : $A = aG$

ECDSA signature

To sign a message M :

1. Generate a random, uniform, secret number $k \in \{1, \ldots, n-1\}$.
2. Compute $(x_1, y_1) = kG$.
3. $r = x_1 \bmod n$.
4. $s = \frac{H(M)+ar}{k} \bmod n$
5. The signature is (r, s) if $r \neq 0$ and $s \neq 0$. Otherwise, restart with a fresh k.

ECDSA verification

We verify a signature (r, s) in the following way :

1. We verify that the public key $A \neq \mathcal{O}$, that A is a point on the curve and that $nA = \mathcal{O}$.
2. We verify that r and s are in $[1, n-1]$.
3. We compute $u_1 = \frac{H(M)}{s} \bmod n$ and $u_2 = \frac{r}{s} \bmod n$.
4. We compute $(x_1, y_1) = u_1 G + u_2 A$
5. We verify that $r = x_1 \bmod n$.

Verification Mistakes

- It is essential to verify the values of r and s.
- April 2022 : CVE-2022-21449, vulnerability in Java 15, 16, 17, 18.
- The check is not done for ECDSA : the signature $(0, 0)$ is always valid !
- Why no division by 0 ? $s^{-1} = s^{n-2}$ by the Little Fermat Theorem when $s \neq 0$.
- Efficient way to compute inverses.

DSA and Randomness

Warning

Both DSA and ECDSA are very **vulnerable** to bad randomness for k. One can **recover the private key**.

- One can recover the key when the **randomness** k **is fixed**.
- One can recover the key when the **randomness** k **depends on the previous randomness** : counter, affine function, ...
- If **few bits** of the randomness k leak (two bits is possible) and hundreds of signatures, one can recover the private key.
- With **one byte** only about 20 signatures are required.
- Very interesting combined with a **software bug** : wrong buffer size, buffer overflow, . . .

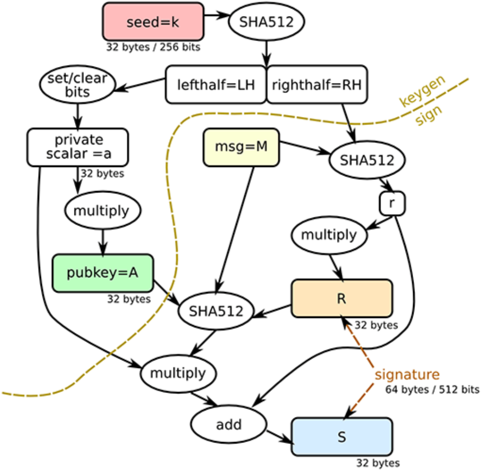
DSA and Security Proof

- There are variants of DSA : **KDSA** and **GDSA**.
- Exist also in elliptic curve variant : **ECKDSA** and **ECGDSA**.
- German and Korean variants.
- The **Korean** variant has a better security proof.
- **Schnorr** signatures are also a better alternative (good proof, simpler equations).
- All these solutions still suffers from randomness problem.

EdDSA

- EdDSA is a **deterministic** signature algorithm.
- Based on **twisted Edward curves**, a special type of elliptic curves with a different equation and different formulas.
- Ed25519 : optimized to be fast on the **x86-64 Nehalem-Westmere** processor family.
- No need of random number generator while signing.
- No branching depending on secrets to avoid side-channel attacks.
- Described in RFC 8032 and introduced in FIPS 186-5.

EdDSA : Global Picture



Source : https://blog.safeheron.com/blog/insights/safeheron-originals/analysis-on-ed25519-use-risks-your-wallet-private-key-can-be-stolen

EdDSA : Parameters and Key Generation

- Let q be a prime number, e.g. $2^{255} - 19$ for Ed25519.
- We are working on an Edward curve E (e.g. Ed25519), with a point B of order ℓ over $\mathrm{GF}(q)$.
- The elliptic curve has $2^c\ell$ points.
- H is SHA-512.
- The **private key** k is a random 256-bit string.
- The **public key** is $A = sB$, where $s = H_{\mathrm{msb}(256)}(k)$, i.e., the 256 most significant bits of the hash.

EdDSA : Signature

- The **signature** of a message M is (R, S), with
- $R = rB$, with $r = H(H_{\text{lsb}(256)}(k) \| M)$
- $S = r + H(R\|A\|M)s \bmod \ell$, with $s = H_{\text{msb}(256)}(k)$.
- The **verification** is $2^c SB = 2^c R + 2^c H(R\|A\|M)A$. (Note that the 2^c is not always needed, see RFC).
- **Implementations mistakes** : API allows to provide A that is different from the public key corresponding to k. Allows to recover k.

Making (EC)-DSA Deterministic

- It is possible to make (EC)-DSA deterministic.
- Formalized in RFC 6979.
- The construction derives the nonce from HMAC-DRBG, the private key and the message.
- Gained a lot of popularity.

Minerva Attack

- 2020 : **Minerva attack** : side-channel attack breaking deterministic (EC)-DSA.
- https://minerva.crocs.fi.muni.cz/
- **Timing attack** allows to recover the **nonce bit-length**.
- Sufficient to recover the **private key**.
- Broke many smart cards and cryptographic libraries.

Implementation Point

It is very hard **not** to leak the bit-length. EdDSA seem to avoid this problem because we take SHA hash which is **not** modulo the order of the curve.

Signature Recommendations

Primitive	Legacy	Future	Remark
RSA-PKCS#1 v1.5	✓	x	no proof
RSA-PSS	✓	✓	Big sizes
(EC)-DSA	✓	x	randomness danger
(EC)-GDSA	✓	x	randomness danger
(EC)-KDSA	✓	✓	randomness danger
(EC)-Schnorr	✓	✓	randomness danger
EdDSA	✓	✓	deterministic

source of recommendations (except last) :

https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf