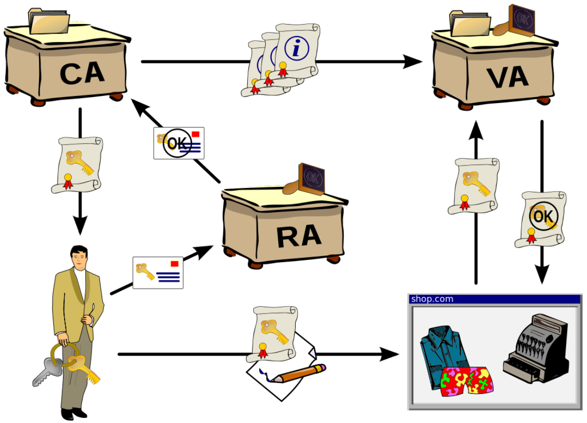
CS4236 Cryptography Theory and Practice Topic 10 - Public keys and RSA

Hugh Anderson

National University of Singapore
School of Computing

October, 2022

Public key systems...



Outline

1. **Asymmetric (public key) schemes, and RSA**
   - Public key encryption and KEM systems
   - RSA (Textbook and practical)
   - Attacks on RSA: factorization
   - Other attacks on RSA

The story so far ... where are we?

The last 9 weeks: weekly steps we have taken

1. We had a historical/contextual introduction in Session 1.
2. We progressed from perfect *secrecy* to perfect *indistinguishability*.
3. *Perfectly* indistinguishable was relaxed to give *computationally* indistinguishable. An EAV-Secure system was constructed with a PRG.
4. The notion of CPA-secure was developed, and achieved with a PRF.
5. Modes, the “padding oracle”, and CCA-Security were outlined.
6. We looked at cryptographic MACs and *authenticated* encryption.
7. We began looking at hashes, with definitions, games, and applications.
8. We then looked at birthday attacks on hashes, and rainbow tables.
9. Last week, we looked at the core of modern symmetric systems - the SP networks. We looked at differential and linear cryptanalysis of such encryption schemes. We saw a bit more of the math needed for Asymmetric cryptography (where we are headed today).

The math ecosystem in this course

A reminder and revision of the pathway followed

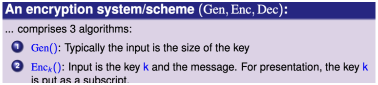
At the beginning of this course, we looked at Shannon's idea of "Perfect" secrecy, where he used probabilities to express the idea that a good cryptosystem would be one where knowing the ciphertext should not help you learn the plaintext. Later we re-formatted this idea into an adversarial game with "Perfect" indistinguishability, because (you were told) this would be useful for other crypto-related things, not just encryption.

The next step involved recognizing the practical situation and relaxing perfection to "Computational" indistinguishability. Most (but not all) of the work since then has been in this *weakened* world. The math framework involves the following elements:

- **Systems/schemes** which are the basic elements of the crypto world.
- **Security properties** over those schemes.
- **Games/experiments** used to define properties.
- **Constructions**, which are instances of schemes, and
- **Proofs**, which may show a property implies another property, or that a construction has a property.

Systems/Schemes

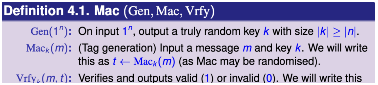
Only 4 schemes in crypto to date. 2 new ones today...



An encryption system/scheme (Gen, Enc, Dec)**:**

... comprises 3 algorithms:

1. Gen(): Typically the input is the size of the key
2. Enc$_k$(): Input is the key k and the message. For presentation, the key k is put as a subscript.

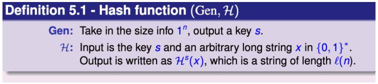


Definition 4.1. Mac (Gen, Mac, Vrfy)

Gen(1^n): On input 1^n, output a truly random key k with size $|k| \geq |n|$.

Mac$_k(m)$: (Tag generation) Input a message m and key k. We will write this as $t \leftarrow$ Mac$_k(m)$ (as Mac may be randomised).

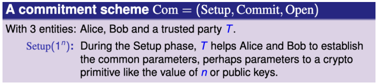
Vrfy$_k(m, t)$: Verifies and outputs valid (1) or invalid (0). We will write this



Definition 5.1 - Hash function (Gen, $\mathcal{H}$)

Gen: Take in the size info 1^n, output a key s.

$\mathcal{H}$: Input is the key s and an arbitrary long string x in $\{0, 1\}^*$. Output is written as $\mathcal{H}^s(x)$, which is a string of length $\ell(n)$.



A commitment scheme Com = (Setup, Commit, Open)

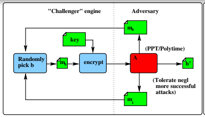
With 3 entities: Alice, Bob and a trusted party T.

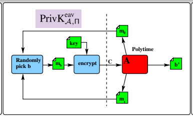
Setup(1^n): During the Setup phase, T helps Alice and Bob to establish the common parameters, perhaps parameters to a crypto primitive like the value of n or public keys.

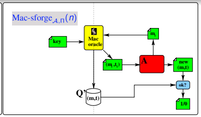
We have had only four systems/schemes Π so far, ENC, MAC, HASH and COM. Note that they just show the overall shape of the scheme, they are not coded, implemented, programmable.

Even so, we define desirable security properties of these schemes, using the game-theoretic approach. This is done before we construct actual working Encryptions, Macs, Hashes or Commitment implementations.

Security properties and games/experiments







You do not need to implement, to state what is desirable

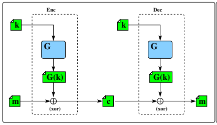
For each of the four schemes, we defined interesting security properties:

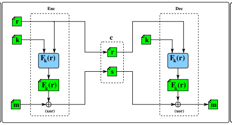
ENC:	EAV, CPA, CCA, unforgeability
MAC:	existential unforgeability, strongly-secure
HASH:	collision resistant
COM:	binding, hiding (together secure)

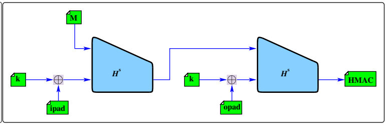
These properties are defined in the context of a specific named (probabilistic) adversarial game, where our adversary $\mathcal{A}$ has some capabilities (an oracle or something), and may be PPT. We name the games as $\mathrm{Name}^{\mathrm{context}}_{\mathcal{A},\Pi}(\mathrm{param})$, where Π is the system, param is a security parameter, Name is just a name for the game, and context identifies the context/capabilities of the adversary.

So far we have had perfect $(= \frac{1}{2})$ and computational $(\leq \mathrm{negl})$, but sometimes you might consider weaker relations: $(\leq)$ or $(\leq \frac{1}{n})$.

Constructions and proofs







ENC, MAC, HASH, COM constructions

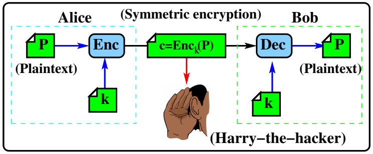
There were a series of constructions, clearly not all the ones in the textbook, but a few of each. We had constructions for

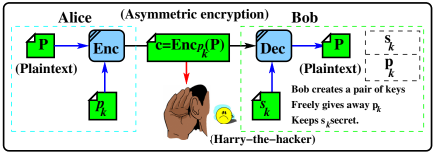
ENC:	3.17 fixed length EAV using PRG, 3.30 CPA from PRF, 4.18 Enc-then-auth
MAC:	4.5 fixed size using PRF, 4.7 variable length, 4.11 CBC-MAC
HASH:	5.3 Merkle Damgård
COM:	(informally) showed a construction based on hashes

Two types of proofs:

1. Prove relationships between security properties.
2. Prove constructions meet some security properties.

Scheme #1: Asymmetric (public key) encryption...





Definition 11.1: $\mathrm{ENC}_{p_k} = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$

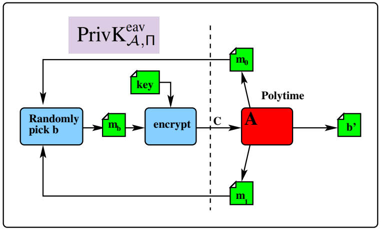
... comprises 3 algorithms:

- $\mathrm{Gen}(1^n)$: The input is the size of the key, returns $\langle p_k, s_k \rangle$.
- $\mathrm{Enc}_{p_k}(m)$: Input is the key p_k and the message.
- $\mathrm{Dec}_{s_k}(c)$: Input is the key s_k and the ciphertext.

As before, for correctness, for all m, we should have: $\mathrm{Dec}_{s_k}(\mathrm{Enc}_{p_k}(m)) = m$

Fine-print: The textbook allows for mistakes in encryption & decryption with negligible probability (The Rabin cryptosystem might be an example of this). In some application scenarios, this might not be acceptable. This is not crucial - the definition probably is formulated in this way to include some known constructions that make negligible mistakes.

Private-key game and definition: $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi}$



Game definition elements - the adversary chooses two

... messages : m_0 and m_1. The challenger then chooses b, and encrypts, to obtain

$$C = \mathrm{Enc}_k(m_b)$$

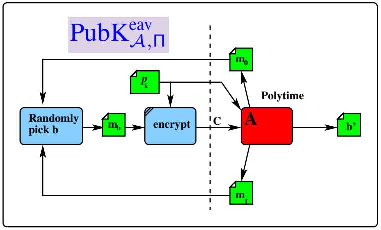
The adversary succeeds if $b' = b$, and output is $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi} = 1$

Definition 3.8 - EAV-secure for private-key schemes

A private-key encryption scheme Π has indistinguishable encryption in the presence of an eav if, for any $\mathcal{A}$, there is a negl, s.t.

$$\Pr[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathrm{negl}(n)$$

Public-key game and definition: $\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A},\Pi}$



Game definition elements - the adversary chooses two

... messages : m_0 and m_1. The challenger then chooses b, and encrypts, to obtain

$$C = \mathrm{Enc}_{p_k}(m_b)$$

The adversary has p_k and succeeds if $b' = b$, and output is $\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A},\Pi} = 1$

Definition 11.2 - EAV-secure for public-key schemes

A public-key encryption scheme Π has indistinguishable encryption in the presence of an eav if, for any $\mathcal{A}$, there is a negl, s.t.

$$\Pr[\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathrm{negl}(n)$$

Public-key $\langle p_k, s_k \rangle$, and private-key (k) systems

EAV, Multi, CPA and CCA are different

As we did before for symmetric schemes, we can define EAV security, Multi-encryption security, CPA security and CCA security for public key encryption, and the corresponding indistinguishability experiments - for example $\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n)$, $\mathrm{PubK}^{\mathrm{LR-cpa}}_{\mathcal{A},\Pi}(n)$ and $\mathrm{PubK}^{\mathrm{cca}}_{\mathcal{A},\Pi}(n)$. Only one was shown here...

There are however significant differences. For public key systems, the encryption oracle is redundant. The public/adversary already can encrypt, and so giving the adversary the encryption oracle will not make any difference. So, if a scheme is EAV secure, it *automatically* implies that the scheme is CPA-secure.

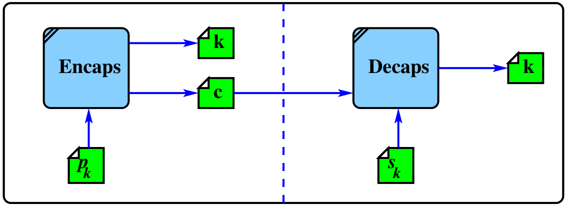
In the textbook, various theorems explore the relationships:

- No deterministic public key encryption is CPA secure (Theorem 11.4).
- If a public key scheme is CPA-secure, then it is also multi-encryption secure (Theorem 11.6).

Perfect public-keys?

It is not possible to have *perfectly secure* public key encryption.

Scheme #2: KEM - Key Encapsulation Mechanism



Definition 11.9: The scheme $\mathrm{KEM}_{p_k} = (\mathrm{Gen}, \mathrm{Encaps}, \mathrm{Decaps})$

... comprises 3 algorithms:

$\mathrm{Gen}(1^n)$**:** The input is the size of the key, returns $\langle p_k, s_k \rangle$.

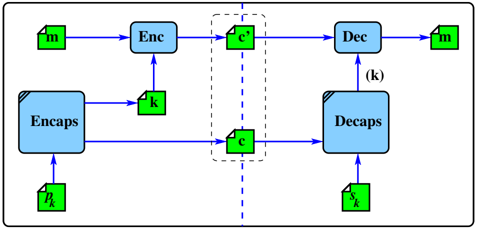
$\mathrm{Encaps}_{p_k}(1^n)$**:** Outputs a key k, and ciphertext: $\langle c, k \rangle \leftarrow \mathrm{Encaps}_{p_k}(1^n)$

$\mathrm{Decaps}_{s_k}(c)$**:** Input is the encrypted ciphertext, output is: $k \leftarrow \mathrm{Decaps}_{s_k}(c)$

As before, for correctness, if $\langle c, k \rangle \leftarrow \mathrm{Encaps}_{p_k}(1^n)$ then $k \leftarrow \mathrm{Decaps}_{s_k}(c)$.

Encaps and Decaps are implemented using Enc_{p_k} and Dec_{s_k} defined before.

Construction #1: Hybrid public key encryption



Construction 11.10: A KEM with symmetric scheme Π'

To encrypt a message m, with the public key p_k, using $\Pi' = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$:

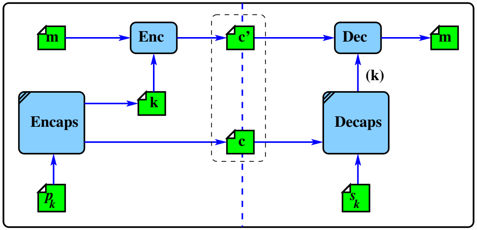
$\mathrm{Gen}^{\mathrm{hy}}(1^n)$: The input is the size of the key, returns $\langle p_k, s_k \rangle$.

$\mathrm{Enc}^{\mathrm{hy}}_{p_k}(m)$: Do $\langle c, k \rangle \leftarrow \mathrm{Encaps}_{p_k}(1^n)$, $c' \leftarrow \mathrm{Enc}_k(m)$. Output: $\langle c, c' \rangle$

$\mathrm{Dec}^{\mathrm{hy}}_{s_k}(c)$: Compute $k \leftarrow \mathrm{Decaps}_{s_k}(c)$ and then output: $m \leftarrow \mathrm{Dec}_k(c')$

Although KEM uses public key encryption, its security requirement is “weaker”. Note that in KEM, instead of encrypting a given message, it encrypts a uniformly selected key. Hence, in the experiment, the adversary can’t decide the messages to be encrypted. As a result, intuitively, it is easier to design a secure KEM compared to a secure public key encryption.

Theorems for this construction



We can define CPA and CCA based on the KEM

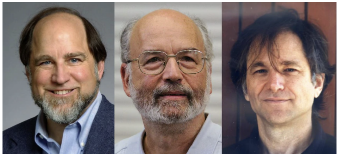
The construction is just using a standard symmetric key system as seen before, so - not too interesting. Instead we make our theorems dependant on the KEM.

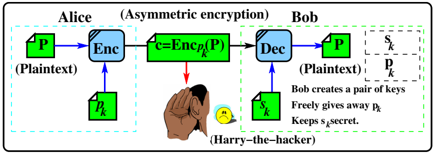
Theorem 11.12: If a KEM is CPA-secure, and Π' is EAV-secure, then Construction 11.10 is a CPA-secure public key encryption.

Theorem 11.14: If a KEM is CCA-secure, and Π' is CCA-secure, then Construction 11.10 is a CCA-secure public key encryption.

We omit the details.

Asymmetric construction #2: RSA





RSA is a well known public key encryption technique

This public key system exploits the difficult problem of trying to find the complete factorization of a large composite integer whose prime factors are not known.

RSA can be used for encryption, digital signatures, and even key exchange.

Keys $\mathrm{GenRSA}(1^n)$, and *textbook* RSA $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$

GenRSA: creates public and private/secret keys p_k and s_k

1. Select two large primes p and q, and assign $N = p \times q$.
2. Compute $\phi(N) = (p-1)(q-1)$.
3. Choose $e > 1$ relative prime to $\phi(N)$.
4. Choose d s.t. $d \times e \bmod \phi(N) = 1$ (i.e. multiplicative inverse).
5. Return $\langle N, e, d\rangle$. $p_k = \langle N, e\rangle$, $s_k = \langle N, d\rangle$.

Construction 11.26: (asymmetric) "textbook" RSA

To encrypt and decrypt a message m:

$\mathrm{Gen}(1^n)$: Compute $\langle N, e, d\rangle \leftarrow \mathrm{GenRSA}(1^n)$, and then set $s_k = \langle N, d\rangle$ and public key $p_k = \langle N, e\rangle$.

$\mathrm{Enc}_{p_k}(m)$: Given $\langle N, e\rangle$, message $m \in \mathbb{Z}_N^*$, compute: $c \leftarrow m^e \bmod N$

$\mathrm{Dec}_{s_k}(c)$: Given $\langle N, d\rangle$, ciphertext c, compute: $m \leftarrow c^d \bmod N$

Correctness and properties

Why does this work?

$$\begin{aligned} c^d \bmod N &= m^{ed} \bmod N \\ &= m^{k(p-1)(q-1)+1} \bmod pq \\ &= m \times m^{k(p-1)(q-1)} \bmod pq \\ &= m \end{aligned}$$

(See the earlier material on Euler's theorem if you cannot follow this).

Textbook RSA has some interesting properties

It is homomorphic w.r.t. multiplication - (this limits it in practice):

$$\mathrm{Enc}_{p_k}(m_1 \times m_2) = \mathrm{Enc}_{p_k}(m_1) \times \mathrm{Enc}_{p_k}(m_2) \bmod N$$

The above property is often called *Partially Homomorphic*, to distinguish it from *Fully Homomorphic*, that is, homomorphic w.r.t. both $+, \times$ in the ring.

“Textbook” RSA properties

Textbook RSA is not CCA-secure

Imagine you had a CCA-style decryption oracle. It will return a plaintext given a previously unseen ciphertext. What can be done with RSA?

Suppose you had a ciphertext $c = m^e \bmod N$, then you could select an integer s, and ask the oracle to decrypt the (previously unseen) ciphertext $(s^e \times c) \bmod N$. It will return $s \times m$, and you can divide by s and discover m. Clearly textbook RSA is not CCA-secure, and this led to various *random* padding constructions.

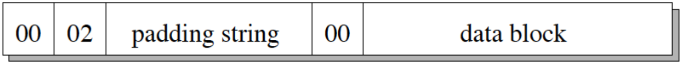
Textbook RSA not even EAV secure

Since textbook RSA is deterministic, it is not even EAV-secure.

Nevertheless, based on it, we can construct CPA-secure and CCA-secure encryptions via “padding”, for example the standard RSA PKCS#1 (RFC2313). Note that the older version v1.5 is not CPA-secure (see pg 415). The story is humbling, both because it is a case study in how **not** to do something, but also how **not** to fix the problem afterwards.

A padding attack on RSA

PKCS#1 padding attack on KEM, from 1998



In 1998, Daniel Bleichenbacher from Bell labs published a padding oracle attack on PKCS#1 (and hence SSL V3.0). The attack is based on the fact that RSA is homomorphic with respect to multiplication. In addition, some (web) servers can act as padding oracles.

In SSL, during the initial negotiation, critical keys are sent, encrypted using RSA (a KEM). A padding oracle attack can retrieve this "pre"master secret key, in what has been called the “million message” attack. The details of this attack are found in the paper, but it is not dissimilar to the padding oracle attack in Session 5; multiple messages are sent to the server, which responds differently if the format of the padding is correct or incorrect.

“Fixing” PKCS#1

The approach taken to fix this was to ask vendors to add extra countermeasures: primarily returning uninformative server responses.

Revisiting PKCS#1

KE/KEM in 2018: RSA, (ECC) DH, (ECC) Ephemeral DH

In 2018, Hanno Böck and others decided to see what the (RSA) situation was after 20 years. They were astonished that it did not seem to be fixed. They managed to identify that Paypal, Apple, Ebay, Facebook, Cisco and others were all vulnerable to Bleichenbacher's attack!

It gets worse...

1. Hanno managed to use the padding oracle to sign a message of their own using Facebook's private key. They did not retrieve the key, but could get the server to sign a message for them!
2. They found that some of the biggest vendors of web systems (for example Cisco ACE) still provided insecure RSA-based TLS. Cisco ACE ONLY provides RSA, and will not update it - it is out of support.

Hmmmmmm...

Standards bodies in 2022

In 2022, all NIST/ISO/IEC standards are driven by formal (mathematical) analysis along the lines of this course. Crypto has to be *proved* correct.

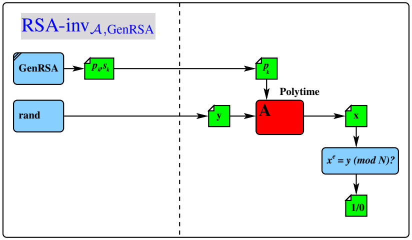
Games/experiments for security of RSA

The RSA problem

Here is one way to formulate the security: consider an experiment where a ciphertext y is uniformly chosen, and the adversary wins if the adversary can decrypt y. (i.e. find an x, s.t. $\mathrm{Enc}_{p_k}(x) = y$). Next, define it to be secure (or the problem to be hard) if the adversary wins with at most negligible probability. This is essentially the RSA problem.

The security of all RSA-based constructions rely on an almost self-fulfilling hardness assumption: that the RSA problem is hard! It is not clear that RSA is secure because factorization is hard. It is easy to show that factorization is a necessary condition for RSA to be secure. However, there is no proof on the sufficient condition side of the argument. The security of RSA is a self-fulfilling statement: an RSA based construction is secure, on the assumption that breaking RSA is hard!

The RSA problem as an experiment



The game: $\text{RSA-inv}_{\mathcal{A},\text{GenRSA}}(n)$ (RSA hard)

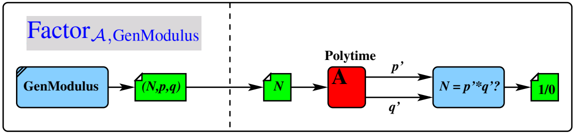
1. Generate keys $\langle N, e, d\rangle \longleftarrow \text{GenRSA}(1^n)$.
2. Uniformly choose y from $\mathbb{Z}_N^*$, and pass to the adversary along with $p_k = \langle N, e\rangle$. Adversary outputs x.
3. Adversary wins (output of experiment is 1) iff $x^e = y \bmod N$.

Definition 8.46 - the “RSA problem”

RSA is hard relative to GenRSA if for all ppt $\mathcal{A}$, there is a negl, s.t.

$$\Pr[\text{RSA-inv}_{\mathcal{A},\text{GenRSA}}(n) = 1] \leq \text{negl}(n)$$

The Factorization problem as an experiment



The game: $\text{Factor}_{\mathcal{A},\text{GenModulus}}(n)$

An assumption here is that we have an algorithm $\text{GenModulus}(1^n)$ which can find two n-bit primes and multiply them together.

1. Generate $\langle N, p, q\rangle$ using $\text{GenModulus}(1^n)$ where $N = p \times q$.
2. Adversary is given N. Adversary outputs p', q', and wins (output of experiment is 1) iff $N = p' \times q'$.

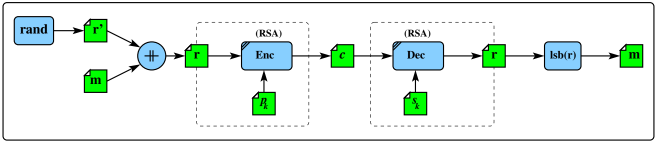
Definition 8.45 - the "Factor problem"

Factoring is hard relative to GenModulus if for all ppt $\mathcal{A}$, there is a negl, s.t.

$$\Pr[\text{Factor}_{\mathcal{A},\text{GenModulus}}(n) = 1] \le \text{negl}(n)$$

The RSA problem being hard implies that factorisation is hard. The other direction is not clear.

Construction #3: CPA security of RSA for 1 bit



Construction 11.32: 1-bit encryption using textbook RSA

Here is a construction that can be proven to be CPA-secure with the "standard model". However, it is not practical: It encrypts only one bit.

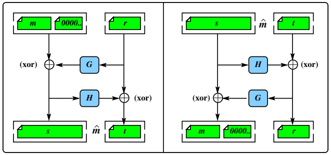
$\mathrm{Gen}(1^n)$: as before

$\mathrm{Enc}_{p_k}(m)$: Given $p_k = \langle N, e \rangle$ and input bit m, uniformly choose r in $\mathbb{Z}_N^*$. Set the last bit of r to m. Output: $c \leftarrow r^e \bmod N$

$\mathrm{Dec}_{s_k}(c)$: Given $s_k = \langle N, d \rangle$, and the ciphertext c, compute $r := c^d \bmod N$. Output the last bit of r.

Another construction (11.34) extends the above (using the idea of KEM) to longer messages. However, it is still not efficient enough to be practical.

Construction #4: OAEP



11.36: Optimal Asymmetric Encryption Padding

The core of this is the idea that before encrypting the message m, it is transformed using a two stage Feistel (DES?) network. It is CCA-secure under the random oracle model.

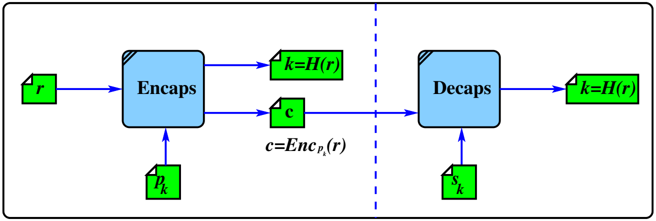
The construction is similar to before, and we just show Enc and Dec:

$\mathrm{Enc}_{p_k}(m)$: on input $m, p_k = \langle N, e\rangle$, append zeros to the message m giving m'. Uniformly choose r and compute $s = m' \oplus G(r)$, $t = r \oplus H(s)$ and $\widehat{m} := s \mathbin{+\!\!+} t$. Output: $c \leftarrow \widehat{m}^e \bmod N$

$\mathrm{Dec}_{s_k}(c)$: Given $s_k = \langle N, d\rangle$, and ciphertext $c \in \mathbb{Z}_N^*$, compute $\widehat{m} := c^d \bmod N$, and invert operations to restore m.

As we saw, textbook RSA is insecure, and it is recommended to use OAEP.

Construction #5: Textbook RSA for secure KEM



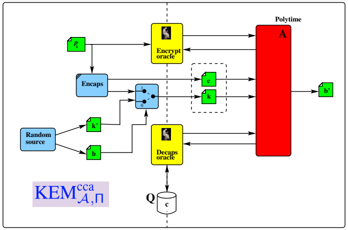
Many practical and provably CCA/CPA-secure schemes employ hashes, but we can only prove their security based on the random oracle assumption.

Construction 11.37, KEM based on textbook RSA

$\mathrm{Gen}(1^n)$:	as before, but additionally specify hash $\mathcal{H} : \mathbb{Z}_N^* \to \{0,1\}^n$
$\mathrm{Encaps}_{p_k}(1^n)$:	on input $p_k = \langle N, e\rangle$ uniformly choose r in $\mathbb{Z}_N^*$. Determine $k := \mathcal{H}(r)$ and output: $c \leftarrow r^e \bmod N$
$\mathrm{Decaps}_{s_k}(c)$:	Given $s_k = \langle N, d\rangle$, and the ciphertext $c \in \mathbb{Z}_N^*$, compute $r := c^d \bmod N$ and output: $k \leftarrow \mathcal{H}(r)$

The message is encrypted using a CCA-secure symmetric scheme, key k.

CCA KEM game and definition: $\mathrm{KEM}^{\mathrm{cca}}_{\mathcal{A},\Pi}$



CCA Game definition elements

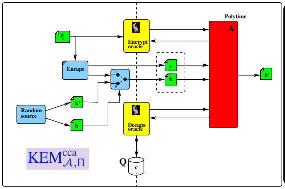
The challenger chooses b, and either returns the current key k, or a uniformly chosen one, along with the ciphertext c. The adversary has p_k and succeeds if $b' = b$, where the output is $\mathrm{KEM}^{\mathrm{cca}}_{\mathcal{A},\Pi} = 1$. Note that this game does not involve any idea of what is encrypted.

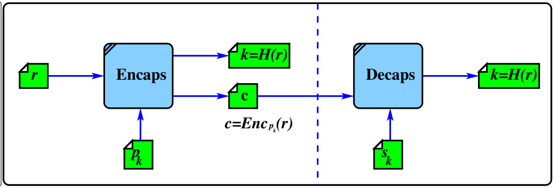
Definition 11.13 - CCA-secure KEM

A key encapsulation mechanism Π is CCA-secure if, for all PPT $\mathcal{A}$, there is a negl, s.t.

$$\Pr[\mathrm{KEM}^{\mathrm{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathrm{negl}(n)$$

CCA KEM game, and RSA, and $\mathcal{H}(r)$





The KEM construction 11.37 based on textbook RSA

In the construction based on textbook RSA, the CCA secureness cannot come from RSA, which we know is not at all secure. However, the test is not to do with decrypting RSA - we do not have an RSA decryption oracle.

Instead we have a decapsulation oracle. This oracle returns the key (if not seen before), rather than the decrypted ciphertext. Given the random oracle model used for the hash function $\mathcal{H}()$, we learn no information from knowing a key.

The factorization problem

State of the art factorization

A straightforward factorization algorithm would take $\mathcal{O}(N^{\frac{1}{2}})$ time, trying all numbers up to the square root of N. But there are many approaches: see `http://en.wikipedia.org/wiki/Integer_factorization`

RSA challenge: `https://en.wikipedia.org/wiki/RSA_numbers`
Very few RSA-encrypted messages have been cracked, and these efforts each only cracked a single RSA key. The current largest keysize that has been cracked was a 829 bits number (RSA-250) that was factored in Feb 2020:

2140324650240744961264423072839333563008614715144755017797754920881418023447
1401366433455190958046796109928518724709145876873962619215573630474547705208
0511905649310668769159001975940569345745223058932597669747168173806936489469
9871578494975937497937
=
6413528947707158027879019017057738908482501474294344720811685963202453234463
0238623598752668347708737661925585694639798853367 *
3337202759497815655622601060535511422794076034476755466678452098702384172921
0037080257448673296881877565718986258036932062711

The Pollard p-1 algorithm

What if all prime factors of p-1 are small?

Factorization Algorithm: Pollard p-1 Algorithm

The algorithm takes in input N, and a value B.
It succeeds when all the factors of $p - 1$ are smaller than B and unique. The running time of the algorithm is polynomial w.r.t. B, and hence this is feasible if B is small.

Why it is worth mentioning this algorithm?

> *A program may choose a new prime p by multiplying the set of primes smaller than some number (say 18): $2 \times 3 \times 5 \times \ldots \times 17$ and then adding 1. Such a method of constructing a prime is vulnerable to the attack above.*

To avoid this attack, choose a prime p s.t. $p - 1$ contains a large factor.

The Pollard rho (ρ) algorithm for factorization

Pollard Rho Algorithm - a birthday attack!

As we saw before, a straightforward factorization algorithm takes $\mathcal{O}(N^{\frac{1}{2}})$ time. By contrast, this algorithm takes an expected $\mathcal{O}(N^{\frac{1}{4}})$ time[a]. Interestingly, it uses a birthday attack, such as we looked at a few weeks ago. Just for curiosity, let's look at the details.

[a]If N is 200 bits, then the security is about 50 bits under this attack.

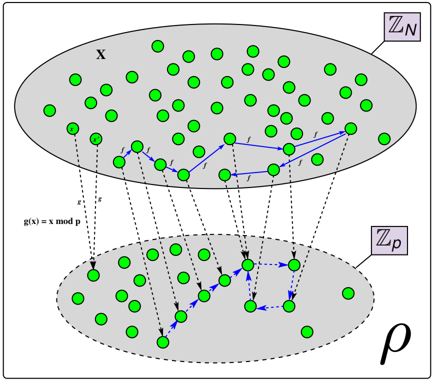
Pollard Rho Algorithm – improvement to $\mathcal{O}(N^{\frac{1}{4}})$

Given $N = p \times q$, and where $p < q$, an important observation is that for any x, x' s.t. (a) $x \neq x'$ and (b) $x = x' \mod p$, then $\gcd(x - x', N) = p$. Why?

A birthday attack! If we have a set X of integers (randomly drawn from $\mathbb{Z}_N$) with $|X| \approx 1.177p^{\frac{1}{2}}$, then by the birthday attack, the chance that there are two elements x, x' s.t. (a) $x \neq x'$ and (b) $x = x' \mod p$ is more than 50%. Hence, potentially, we have a $\mathcal{O}(p^{\frac{1}{2}}) \approx \mathcal{O}(N^{\frac{1}{4}})$ algorithm to factorize N.

Now we need an algorithm to find two elements in the set X which have the same reminder modulo p. Hmmmmm... sounds like a collision...

Find the collision fast



Supergroup Floyd returns! With a ρ

Above we see the two sets of integers $\mathbb{Z}_N$ and $\mathbb{Z}_p$. The (somewhat imaginary) matching sequence in $\mathbb{Z}_p$ will cycle more quickly than that in $\mathbb{Z}_N$. We do not know p, but even so, if we have two values x, x' in $\mathbb{Z}_N$, and they both match the same value in $\mathbb{Z}_p$ (modulo p), then we know that $\gcd(x - x', N) = p$.

The function f used in $\mathbb{Z}_N$ is commonly $f(x) = (x^2 + 1) \bmod N$.

The algorithm in python

Algorithm 9.2: Floyd loop comes back!

```
import sys
xinit,N = (int(sys.argv[1]),int(sys.argv[2]))
def gcd(x, y):
   while(y):
      x, y = y, x % y
   return x
def f(x):
   return ((x*x)+1) % N

def floyd(x0):
   t = f(x0)
   h = f(f(x0))
   p = gcd(t-h,N)
   while (p == 1):
      t = f(t)
      h = f(f(h))
      p = gcd(t-h,N)
   if (p == N):
      return -1
   else:
      return p
print("Factor is ", floyd(xinit))
```

Our analysis assumes that the function f gives a *truly random* set, which is not true. Nevertheless, studies show that the running time is in the order of $\mathcal{O}(p^{\frac{1}{2}}) \approx \mathcal{O}(N^{\frac{1}{4}})$. This is still exponential.

Practical factorization

Other factorization algorithms

Other known factoring algorithms, and their corresponding theoretical complexity/runtimes are:

- Elliptic curve: Complexity is $\mathcal{O}\left(e^{\sqrt{(2+o(1))\ln p \ln\ln p}}\right)$

  where p is the smallest factor in N.
- Dixon's: Complexity is $\mathcal{O}\left(e^{2\sqrt{2}\sqrt{\ln N \ln\ln N}}\right)$
- Quadratic sieve: Complexity is $\mathcal{O}\left(e^{(1+o(1))\sqrt{\ln N \ln\ln N}}\right)$
- Number field sieve: Complexity is $\mathcal{O}\left(e^{(1.92+o(1))(\ln N)^{\frac{1}{3}}(\ln\ln N)^{\frac{2}{3}}}\right)$

So, if N is a 512 bit number, the simple exhaustive search requires roughly 2^{256} steps, Pollard-rho perhaps 2^{128} steps. However, number field sieve factoring requires a much smaller number of steps, in the order of 2^{67}.

$o(1)$ is a function of N that approaches 0 when $N \to \infty$.

Weird science - Quantum physics

First interest: Quantum computing...

Quantum *computers* may be able to compute HARD problems quickly (such as factorizing large composites).

How? The underlying data elements are quantum bits (qubits), not limited to just 0,1 states - instead considered to be a superposition of states. An operation performed on a qubit is performed on all the states simultaneously. Shor's algorithm.

DWave systems have sold the first commercial quantum computer, with (evidently) 128 qubits. However, it is unable to perform Shor's algorithm:

`http://www.dwavesys.com/`

It is likely that no effective quantum computer has yet been built that could factor a large composite.

In addition, you might have heard the term quantum communication in reference to confidentiality. This technique identifies when a communication is observed by an adversary, and refuses to continue the communication protocol. I do not think it fits in a crypto course.

Shor's Algorithm

A very slow way to find factors of $p \times q$?

Choose some number a, with no factors[a] in common with pq. Imagine you calculate a^2, a^3, a^4, all modulo pq, until the sequence repeats for the first time (perhaps after r steps). The repetition value r evenly divides $(p-1)(q-1)$. Have a look at the Euler table in slide set 9 - it shows the powers modulo $15 = pq = 3 \times 5$, and all repetitions divide $(p-1)(q-1) = 8$.

In addition $a^r \equiv 1 \bmod pq$, and since this was the first repetition, $b = a^{\frac{r}{2}} \neq 1 \bmod pq$. But, $b^2 \equiv 1 \bmod pq$.

We have four square roots of 1: ± 1 and $\pm b$. If we know b, we can calculate the factors of pq.

[a]Of course if your number a does have factors in common with pq, you have either discovered p or q.

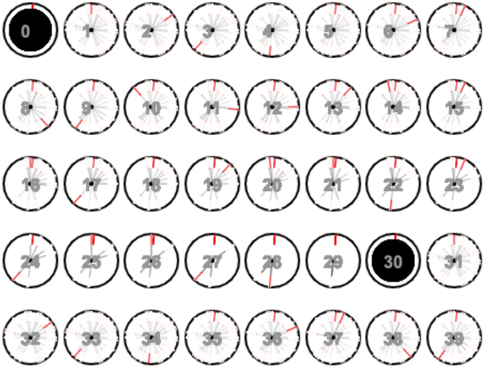
It is slow because...
...the sequences may be very very long. So this is not some secret fast algorithm if you have to calculate long sequences a^2, a^3, a^4, all modulo pq.

Shor’s Algorithm

Find repetition values for all powers simultaneously:

A Quantum register holds all possible values at the same time, and we perform an amplifying operation, that only leaves stable repetition values.

Mark the same times every day... In the simulation, only the clock with a repetition rate that we are interested in remains:



Post-quantum cryptography

What is post-quantum cryptography?

There are quantum algorithms (like Shor) that can be used to solve most of the current techniques for public-key systems (factorization and the discrete-log problem).

It seems not likely that quantum computers are yet able to solve the bit size of current systems, but still there is interest in devising systems which could not be solved by a quantum algorithm. These quantum-safe systems are based on known NP-hard problems.

Some examples of (hopefully) quantum-safe cryptography are:

- Lattice-based cryptography: Given the “basis” of a lattice, the “Short Integer Solution” problem is known to be hard.
- Multivariate polynomial: Solving systems of multivariate polynomials is known to be hard.
- Code-based cryptography: Decoding some randomised error-correcting codes is known to be hard.

See here: `https://en.wikipedia.org/wiki/Post-quantum_cryptography`

Small exponents: e or d

Small (public) exponent e

If the RSA exponent e is small (for example 3), the encryption $x^3 \bmod N$ can be computed very quickly. In practice, there are many examples of public keys with exponents of $3, 17$ and so on.

Coppersmith's attack can recover some factors if multiple entities use the same e (but different N of course) in their public keys, and then send the same message. The attack can be avoided with proper padding.

The NIST Special Publication on Computer Security (SP 800-78 Rev 4 of May 2015) recommends a public exponent e of $2^{16} + 1 = 65537$, or random (larger) exponents: `http://dx.doi.org/10.6028/NIST.SP.800-78-4`
See NUS's certificate!

Small (private) exponent d

There is an algorithm that can efficiently compute d from (N, e), if d is small (for example, if $d < \frac{1}{3}N^{\frac{1}{4}}$ and $q < p < 2q$).
Wiener, Michael J., "Cryptanalysis of short RSA secret exponents", IEEE Transactions on Information Theory, 36 (3): 553–558, May 1990.

Other approaches?

Finding $\phi(N)$? **(Not close, no cigar)**

If an adversary can find $\phi(N)$ from the public key $\langle N, e\rangle$, the private key d can be computed, and thus break RSA.

The above observation leads to this question: Can an adversary break RSA by finding $\phi(N)$ directly, and thus avoiding factoring N? Fortunately, finding $\phi(N)$ is not easier than factoring N. Suppose Alice knows the value of $\phi(N)$. She can form the following equations with two unknowns p, q:

$$\begin{aligned} N &= p \times q \\ \phi(n) &= (p-1) \times (q-1) \end{aligned}$$

By substituting $q = \frac{N}{p}$ in the second equation, we have

$$p^2 - Ap + N = 0$$

where $A = (N - \phi(N) + 1)$.

Note that the above equation can be solved easily. Finding a square root is easy if we do not have modulo. So if Alice can find $\phi(N)$, then she can factor N. Therefore, finding $\phi(N)$ is not easier than factoring N.

Finding d, m directly maybe?

Finding d directly?

Since factoring N and finding $\phi(N)$ are computationally equivalent, an adversary may want to find d directly from the public key.

There is a deterministic polynomial time algorithm that, given e, d and N, factors N in polynomial time. So finding d directly is not easier than factoring N.

In other words, to find the RSA private key from the public key is as difficult as factoring N.

Finding m directly?

Given c, N, could an adversary find m directly, without the intermediate step of finding the value of d? This is just back to the RSA problem.

Summary on setups/configurations

Given the previous information...

Here is a possible checklist of things to ensure that keys are used correctly:

1. The size of p, q should be almost the same, as the running time of factorization algorithms depend on the size of the smaller prime among p, q. The difference $|p - q|$ should still be large, as an exhaustive search around $\sqrt{N}$ may find the factors.
2. p, q are preferably*strong primes*. A randomly chosen prime most likely is a strong prime. A strong prime p satisfies the following conditions:
   - $p - 1$ has a large prime factor (lets call it r), to avoid the Pollard algorithm.
   - $p + 1$ has a large prime factor, to avoid the $p + 1$ factoring algorithm.
   - $r - 1$ has a large prime factor, to avoid an attack known as the cycling attack.
3. The size of d should be large, due to the attacks on small d.
4. If d is accidentally revealed, the composite N has to be changed.

Leakage from ciphertext

RSA leaks a bit...

The ciphertext of RSA leaks one “bit” of information. From the ciphertext $c = m^e \bmod N$, it is possible to extract a small amount of information about the message m.

The Jacobi symbol introduced previously can be efficiently computed, and it takes the value of $0, -1$ or 1. The Jacobi symbol has the following property (inherited from the Legendre symbol):

$$\left(\frac{xy}{N}\right) = \left(\frac{x}{N}\right)\left(\frac{y}{N}\right)$$

And so if e is odd, the Jacobi symbol of c is the same as the Jacobi symbol of m, and as a result, from c, N, the adversary knows a small amount of information about m.

RSA attack summary

Summary of possible attack methods

- Brute force key search - mostly infeasible given the size of numbers.
- Mathematical attacks - based on computing $\phi(N)$, or factoring the modulus N.
- Ciphertext attacks - given properties of RSA.
- Timing/side-channel attacks - Approaches to do timing analysis of the algorithms used were developed by Paul Kocher in the mid-1990's. They exploit timing variations in operations, and countermeasures include techniques to use constant exponentiation time, to add random delays, and to blind values used in calculations. See next week.

Symmetric encryption schemes

$\mathrm{ENC}_k = (\mathrm{Gen}(1^n), \mathrm{Enc}_k(m), \mathrm{Dec}_k(c))$ **(ch1)**

Properties	Defs	Constructions	Proofs
Perfect secrecy	2.3	(One time pad)	Thm 2.9
$\updownarrow$			Lemma 2.6
Perfect indistinguishability	2.5		
$\downarrow$			
EAV-secure	3.8	3.17 (PRG)	Thm 3.18
$\uparrow$			Proof given
EAV-Multi-encryption	3.19		
$\uparrow$			
CPA-Multi-encryption	3.22		
$\updownarrow$			Thm 3.24
CPA-secure	3.23	3.30 (PRF)	Thm 3.31
$\uparrow$			
CCA-secure	3.33		
$+$			
Unforgeable	4.16		
$\updownarrow$			Thm 4.19
Authenticated	4.17	4.18 (Enc-then-Auth)	

MAC and HASH schemes

$$\mathrm{MAC}_k = (\mathrm{Gen}(1^n), \mathrm{Mac}_k(m), \mathrm{Vrfy}_k(m,t)) \tag{4.1}$$

Properties	Defs	Constructions	Proofs
Unforgeable	4.2		
$\uparrow$			Discussed
Strong	4.3	4.5 (Using PRF)	Thm 4.5
		4.7 (Variable length)	Thm 4.8
		4.11 (CBC-MAC)	Thm 4.12
		4.11(a) (Variable CBC-MAC)	

$$\mathrm{HASH} = (\mathrm{Gen}(1^n), \mathcal{H}^s(x)) \tag{5.1}$$

Properties	Defs	Constructions	Proofs
CollisionResistant	5.2	5.3 (Long message)	

Asymmetric encryption schemes

$$\mathrm{ENC}_{p_k} = (\mathrm{Gen}(1^n), \mathrm{Enc}_{p_k}(m), \mathrm{Dec}_{s_k}(c)) \quad \textbf{(11.1)}$$

Properties	Defs	Constructions	Proofs
		11.26 (Textbook RSA)	
		(Elgamal)	Thm 11.18
		(ECC)	
DDH-hard	8.63		
Factor-hard	8.45		
$\uparrow$			
RSA-hard	8.46		
EAV-secure	11.2		
$\updownarrow$			
CPA-secure		11.32 (1-Bit RSA)	
$\uparrow$			
CCA-secure			

Commitment, KEM schemes

$\mathrm{COM} = (\mathrm{Setup}(1^n), \mathrm{Commit}(a), \mathrm{Open}(c_a))$ **(ch5)**

Properties	Defs	Constructions	Proofs
Secure (Binding+Hiding)	5.13	(Hash based system)	

$\mathrm{KEM}_{p_k} = (\mathrm{Gen}(1^n), \mathrm{Encaps}_{p_k}(1^n), \mathrm{Decaps}_{s_k}(c))$ **(11.9)**

Properties	Defs	Constructions	Proofs
CPA-secure		11.10 (CPA_KEM+EAV_ENC)	Thm 11.12
CCA-secure	11.13	11.10 (CCA_KEM+CCA_ENC)	Thm 11.13
		11.37 (CCA_KEM+TextbookRSA)	