Public-Key Setting

Why study private key cryptography at all?

- If two parties with to communicate, they can always generate two public/secret key pairs instead of a secret shared key
- Public-key schemes are slower
- Public-key schemes require longer keys
- Public-key schemes have larger ciphertext expansion
- Public-key schemes require stronger assumptions

These issues can be mitigated

Encrypting long messages

The public key encryption schemes we will see are defined for short messages

How can we encrypt long (arbitrary length) messages?

Encrypting long messages

The public key encryption schemes we will see are defined for short messages

How can we encrypt long (arbitrary length) messages?

Easy solution: Split the message into small blocks, and encrypt each block separately

Encrypting long messages

The public key encryption schemes we will see are defined for short messages

How can we encrypt long (arbitrary length) messages?

Easy solution: Split the message into small blocks, and encrypt each block separately

$\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$

$m = m_1 \,\|\, m_2 \,\|\, m_3 \,\|\, \ldots$

$\mathsf{Enc}'(m) = \mathsf{Enc}_{pk}(m_1) \,\|\, \mathsf{Enc}_{pk}(m_2) \,\|\mathsf{Enc}_{pk}(m_3) \,\|\, \ldots$

Dec'_{sk} is defined in the obvious way

Encrypting long messages

The public key encryption schemes we will see are defined for short messages

How can we encrypt long (arbitrary length) messages?

Easy solution: Split the message into small blocks, and encrypt each block separately

$\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ $m = m_1 \,\|\, m_2 \,\|\, m_3 \,\|\, \ldots$

$\mathsf{Enc}'(m) = \mathsf{Enc}_{pk}(m_1) \,\|\, \mathsf{Enc}_{pk}(m_2) \,\|\mathsf{Enc}_{pk}(m_3) \,\|\, \ldots$

Dec'_{sk} is defined in the obvious way

If Π is CPA-secure (for short length messages), then the above scheme $(\mathsf{Dec}, \mathsf{Enc}', \mathsf{Dec}')$ is CPA-secure for arbitrary length messages.

Encrypting long messages

The public key encryption schemes we will see are defined for short messages

How can we encrypt long (arbitrary length) messages?

Easy solution: Split the message into small blocks, and encrypt each block separately

$\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$

$m = m_1 \,\|\, m_2 \,\|\, m_3 \,\|\, \ldots$

$\mathsf{Enc}'(m) = \mathsf{Enc}_{pk}(m_1) \,\|\, \mathsf{Enc}_{pk}(m_2) \,\|\mathsf{Enc}_{pk}(m_3) \,\|\, \ldots$

Dec'_{sk} is defined in the obvious way

> If Π is CPA-secure (for short length messages), then the above scheme $(\mathsf{Dec}, \mathsf{Enc}', \mathsf{Dec}')$ is CPA-secure for arbitrary length messages.

Drawbacks:

- Slow

Encrypting long messages

The public key encryption schemes we will see are defined for short messages

How can we encrypt long (arbitrary length) messages?

Easy solution: Split the message into small blocks, and encrypt each block separately

$\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ $\qquad$ $m = m_1 \,\|\, m_2 \,\|\, m_3 \,\|\, \ldots$

$$\mathsf{Enc}'(m) = \mathsf{Enc}_{pk}(m_1) \,\|\, \mathsf{Enc}_{pk}(m_2) \,\|\mathsf{Enc}_{pk}(m_3) \,\|\, \ldots$$

Dec'_{sk} is defined in the obvious way

> If Π is CPA-secure (for short length messages), then the above scheme $(\mathsf{Dec}, \mathsf{Enc}', \mathsf{Dec}')$ is CPA-secure for arbitrary length messages.

Drawbacks:

- Slow
- Ciphertext, expansion

Encrypting long messages

The public key encryption schemes we will see are defined for short messages

How can we encrypt long (arbitrary length) messages?

Easy solution: Split the message into small blocks, and encrypt each block separately

$\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$

$m = m_1 \,\|\, m_2 \,\|\, m_3 \,\|\, \ldots$

$\mathsf{Enc}'(m) = \mathsf{Enc}_{pk}(m_1) \,\|\, \mathsf{Enc}_{pk}(m_2) \,\|\mathsf{Enc}_{pk}(m_3) \,\|\, \ldots$

Dec'_{sk} is defined in the obvious way

> If Π is CPA-secure (for short length messages), then the above scheme $(\mathsf{Dec}, \mathsf{Enc}', \mathsf{Dec}')$ is CPA-secure for arbitrary length messages.

Drawbacks:

- Slow
- Ciphertext, expansion

Can we do better?

Hybrid encryption

Idea:

- Use a public-key encryption scheme to encrypt a key k for a private-key encryption scheme
- Use k to encrypt m
- The ciphertext contains both the encryption of k and the encryption of m

Hybrid encryption

Idea:

- Use a public-key encryption scheme to encrypt a key k for a private-key encryption scheme
- Use k to encrypt m
- The ciphertext contains both the encryption of k and the encryption of m

Note: the resulting scheme is a public-key encryption scheme

- the key k does not need to be shared by the parties beforehand

Hybrid encryption

Idea:

- Use a public-key encryption scheme to encrypt a key k for a private-key encryption scheme
- Use k to encrypt m
- The ciphertext contains both the encryption of k and the encryption of m

Note: the resulting scheme is a public-key encryption scheme

  - the key k does not need to be shared by the parties beforehand

Benefits:

- Fast (public-key encryption is used only for the first block)

Hybrid encryption

Idea:

- Use a public-key encryption scheme to encrypt a key k for a private-key encryption scheme
- Use k to encrypt m
- The ciphertext contains both the encryption of k and the encryption of m

Note: the resulting scheme is a public-key encryption scheme

- the key k does not need to be shared by the parties beforehand

Benefits:

- Fast (public-key encryption is used only for the first block)
- Asymptotically optimal ciphertext expansion: the only blocks that are expanded are the first (due to public-key encryption) and possibly the last (due to padding)

Key Encapsulation Mechanism

We are only using the public-key encryption scheme to encrypt a symmetric secret key

We don't need the fully fledged formal definition of a public-key encryption scheme for such task

The idea is better formalized by the notion of **Key Encapsulation Mechanism** (KEM)

Key Encapsulation Mechanism

We are only using the public-key encryption scheme to encrypt a symmetric secret key

We don't need the fully fledged formal definition of a public-key encryption scheme for such task

The idea is better formalized by the notion of **Key Encapsulation Mechanism** (KEM)

A Key Encapsulation Mechanism consists of three algorithms:

- Gen is a randomized polynomial-time algorithm that takes as input the security parameter 1^n and outputs a public/secret key pair (pk, sk), where pk and sk have length at least n.

Key Encapsulation Mechanism

We are only using the public-key encryption scheme to encrypt a symmetric secret key

We don't need the fully fledged formal definition of a public-key encryption scheme for such task

The idea is better formalized by the notion of **Key Encapsulation Mechanism** (KEM)

A Key Encapsulation Mechanism consists of three algorithms:

- Gen is a randomized polynomial-time algorithm that takes as input the security parameter 1^n and outputs a public/secret key pair (pk, sk), where pk and sk have length at least n.

- Encaps is a randomized polynomial-time **encapsulation** algorithm that takes as input a public key pk and 1^n, and outputs a pair (c, k) where c is a ciphertext and $k \in \{0,1\}^{\ell(n)}$, for some key length $\ell(n)$. We write this as $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$.

Key Encapsulation Mechanism

We are only using the public-key encryption scheme to encrypt a symmetric secret key

We don't need the fully fledged formal definition of a public-key encryption scheme for such task

The idea is better formalized by the notion of **Key Encapsulation Mechanism** (KEM)

A Key Encapsulation Mechanism consists of three algorithms:

- Gen is a randomized polynomial-time algorithm that takes as input the security parameter 1^n and outputs a public/secret key pair (pk, sk), where pk and sk have length at least n.

- Encaps is a randomized polynomial-time **encapsulation** algorithm that takes as input a public key pk and 1^n, and outputs a pair (c, k) where c is a ciphertext and $k \in \{0,1\}^{\ell(n)}$, for some key length $\ell(n)$. We write this as $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$.

- Decaps is a deterministic polynomial-time **decapsulation** algorithm that takes as input a secret key sk and a ciphertext c and outputs a key k or a special symbol $\perp$. We denote an execution of this algorithm by $\mathsf{Decaps}_{sk}(c)$.

Key Encapsulation Mechanism

We are only using the public-key encryption scheme to encrypt a symmetric secret key

We don't need the fully fledged formal definition of a public-key encryption scheme for such task

The idea is better formalized by the notion of **Key Encapsulation Mechanism** (KEM)

A Key Encapsulation Mechanism consists of three algorithms:

- Gen is a randomized polynomial-time algorithm that takes as input the security parameter 1^n and outputs a public/secret key pair (pk, sk), where pk and sk have length at least n.

- Encaps is a randomized polynomial-time **encapsulation** algorithm that takes as input a public key pk and 1^n, and outputs a pair (c, k) where c is a ciphertext and $k \in \{0,1\}^{\ell(n)}$, for some key length $\ell(n)$. We write this as $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$.

- Decaps is a deterministic polynomial-time **decapsulation** algorithm that takes as input a secret key sk and a ciphertext c and outputs a key k or a special symbol $\perp$. We denote an execution of this algorithm by $\mathsf{Decaps}_{sk}(c)$.

Correctness: if $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$ then $\mathsf{Decaps}_{sk}(c) = k$, except for negligible probability.

Key Encapsulation Mechanism

We are only using the public-key encryption scheme to encrypt a symmetric secret key

We don't need the fully fledged formal definition of a public-key encryption scheme for such task

The idea is better formalized by the notion of **Key Encapsulation Mechanism** (KEM)

A Key Encapsulation Mechanism consists of three algorithms:

- Gen is a randomized polynomial-time algorithm that takes as input the security parameter 1^n and outputs a public/secret key pair (pk, sk), where pk and sk have length at least n.

- Encaps is a randomized polynomial-time **encapsulation** algorithm that takes as input a public key pk and 1^n, and outputs a pair (c, k) where c is a ciphertext and $k \in \{0,1\}^{\ell(n)}$, for some key length $\ell(n)$. We write this as $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$.

- Decaps is a deterministic polynomial-time **decapsulation** algorithm that takes as input a secret key sk and a ciphertext c and outputs a key k or a special symbol $\perp$. We denote an execution of this algorithm by $\mathsf{Decaps}_{sk}(c)$.

Correctness: if $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$ then $\mathsf{Decaps}_{sk}(c) = k$, except for negligible probability.

Note: a public-key encryption scheme is *a possible way* to build a KEM

The KEM/DEM Paradigm

Unsurprisingly... we can use a KEM $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ and a private-key encryption scheme $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ to build a public-key encryption scheme $\Pi^{hy} = (\mathsf{Gen}^{hy}, \mathsf{Enc}^{hy}, \mathsf{Dec}^{hy})$.

(that's the whole point of introducing KEMs)

The KEM/DEM Paradigm

Unsurprisingly... we can use a KEM $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ and a private-key encryption scheme $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ to build a public-key encryption scheme $\Pi^{hy} = (\mathsf{Gen}^{hy}, \mathsf{Enc}^{hy}, \mathsf{Dec}^{hy})$.

(that's the whole point of introducing KEMs)

- The private-key encryption scheme Π' is called a **Data Encapsulation Mechanism** (DEM)

The KEM/DEM Paradigm

Unsurprisingly... we can use a KEM $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ and a private-key encryption scheme $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ to build a public-key encryption scheme $\Pi^{hy} = (\mathsf{Gen}^{hy}, \mathsf{Enc}^{hy}, \mathsf{Dec}^{hy})$.

(that's the whole point of introducing KEMs)

- The private-key encryption scheme Π' is called a **Data Encapsulation Mechanism** (DEM)
- Assume that Π' uses keys of length $\ell(n)$, i.e., the keys output Encaps and Gen' are "compatible"

The KEM/DEM Paradigm

Unsurprisingly... we can use a KEM $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ and a private-key encryption scheme $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ to build a public-key encryption scheme $\Pi^{hy} = (\mathsf{Gen}^{hy}, \mathsf{Enc}^{hy}, \mathsf{Dec}^{hy})$.

(that's the whole point of introducing KEMs)

- The private-key encryption scheme Π' is called a **Data Encapsulation Mechanism** (DEM)
- Assume that Π' uses keys of length $\ell(n)$, i.e., the keys output Encaps and Gen' are "compatible"

$\mathsf{Gen}^{hy}(1^n)$: $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$; Return (pk, sk)

The KEM/DEM Paradigm

Unsurprisingly... we can use a KEM $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ and a private-key encryption scheme $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ to build a public-key encryption scheme $\Pi^{hy} = (\mathsf{Gen}^{hy}, \mathsf{Enc}^{hy}, \mathsf{Dec}^{hy})$.

(that's the whole point of introducing KEMs)

- The private-key encryption scheme Π' is called a **Data Encapsulation Mechanism** (DEM)
- Assume that Π' uses keys of length $\ell(n)$, i.e., the keys output Encaps and Gen' are "compatible"

$\mathsf{Gen}^{hy}(1^n)$: $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$; Return (pk, sk)

$\mathsf{Enc}^{hy}_{pk}(m)$:

- $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$
- $c' \leftarrow \mathsf{Enc}'_k(m)$
- Return $c \,\|\, c'$

The KEM/DEM Paradigm

Unsurprisingly... we can use a KEM $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ and a private-key encryption scheme $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ to build a public-key encryption scheme $\Pi^{hy} = (\mathsf{Gen}^{hy}, \mathsf{Enc}^{hy}, \mathsf{Dec}^{hy})$.

(that's the whole point of introducing KEMs)

- The private-key encryption scheme Π' is called a **Data Encapsulation Mechanism** (DEM)
- Assume that Π' uses keys of length $\ell(n)$, i.e., the keys output Encaps and Gen$'$ are "compatible"

$\mathsf{Gen}^{hy}(1^n)$: $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$; Return (pk, sk)

$\mathsf{Enc}^{hy}_{pk}(m)$:

- $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$
- $c' \leftarrow \mathsf{Enc}'_k(m)$
- Return $c \,\|\, c'$

$\mathsf{Dec}^{hy}_{sk}(c'')$:

- Parse c'' as $c\|c'$ (return $\perp$ if this fails)
- $k \leftarrow \mathsf{Decaps}_{sk}(c)$
- $m \leftarrow \mathsf{Dec}'_k(c')$
- Return m

CPA-Security of KEMs

Let $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM and let $\mathcal{A}$ be an algorithm. We denote the following experiment by $\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}$

- A key pair $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ is generated
- $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$
- A uniform random bit $b \in \{0, 1\}$ is generated
- If $b = 0$, we set $\widehat{k} = k$. Otherwise ($b = 1$) we choose $\widehat{k}$ u.a.r. from $\{0, 1\}^{\ell(n)}$
- The values of pk, c, and $\widehat{k}$ are sent to $\mathcal{A}$
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ about b
- The *outcome of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

CPA-Security of KEMs

Let $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM and let $\mathcal{A}$ be an algorithm. We denote the following experiment by $\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}$

- A key pair $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ is generated
- $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$
- A uniform random bit $b \in \{0, 1\}$ is generated
- If $b = 0$, we set $\widehat{k} = k$. Otherwise $(b = 1)$ we choose $\widehat{k}$ u.a.r. from $\{0, 1\}^{\ell(n)}$
- The values of pk, c, and $\widehat{k}$ are sent to $\mathcal{A}$
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ about b
- The *outcome of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

Definition: *A key-encapsulation mechanism Π is* **CPA-secure** *if, for every probabilistic polynomial-time adversary $\mathcal{A}$, there is a negligible function ε such that:*

$$\Pr[\mathit{KEM}^{\mathit{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$$

CPA-Security of Hybrid encryption

Theorem: *If Π is a CPA-secure KEM and Π' is an EAV-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CPA-secure public-key encryption scheme.*

CPA-Security of Hybrid encryption

Theorem: *If Π is a CPA-secure KEM and Π' is an EAV-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CPA-secure public-key encryption scheme.*

Notice that only EAV-security is needed for Π' !

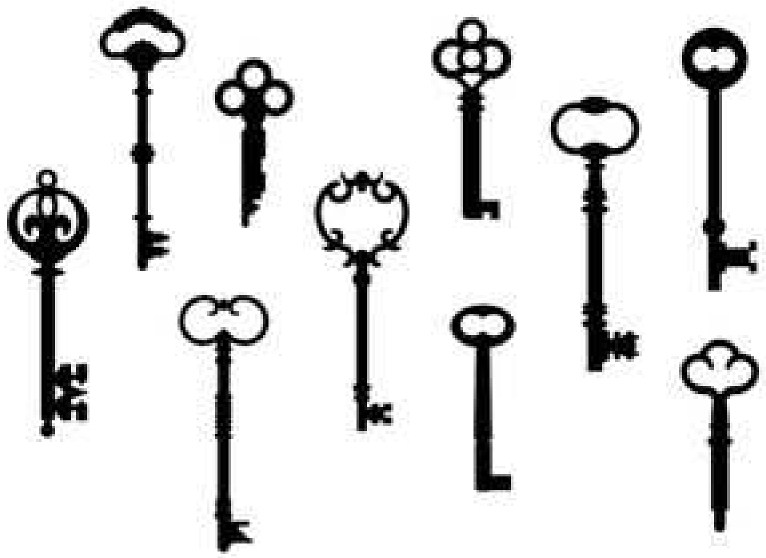
Why?

CPA-Security of Hybrid encryption

Theorem: *If Π is a CPA-secure KEM and Π' is an EAV-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CPA-secure public-key encryption scheme.*

Notice that only EAV-security is needed for Π' ! **Why?**

- A new key k is used for each encryption!



CPA-Security of Hybrid encryption

Theorem: *If Π is a CPA-secure KEM and Π' is an EAV-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CPA-secure public-key encryption scheme.*

High-level idea of the proof:

- Let k^* be a key chosen u.a.r. from $\{0,1\}^{\ell(n)}$
- Let $\approx$ denote the relation "being indistinguishable by a polynomial-time adversary" defined over probability distributions
- Notice that $\approx$ is reflexive, symmetric, and transitive when applied a polynomial number of times

CPA-Security of Hybrid encryption

Theorem: *If Π is a CPA-secure KEM and Π' is an EAV-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CPA-secure public-key encryption scheme.*

High-level idea of the proof:

- Let k^* be a key chosen u.a.r. from $\{0,1\}^{\ell(n)}$
- Let $\approx$ denote the relation "being indistinguishable by a polynomial-time adversary" defined over probability distributions
- Notice that $\approx$ is reflexive, symmetric, and transitive when applied a polynomial number of times

$$(pk, \mathsf{Enc}^{hy}_{pk}(m_0)) \quad \underset{\text{we want this}}{\overset{\approx}{\longleftrightarrow}} \quad (pk, \mathsf{Enc}^{hy}_{pk}(m_1))$$

CPA-Security of Hybrid encryption

Theorem: *If Π is a CPA-secure KEM and Π' is an EAV-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CPA-secure public-key encryption scheme.*

High-level idea of the proof:

- Let k^* be a key chosen u.a.r. from $\{0,1\}^{\ell(n)}$
- Let $\approx$ denote the relation "being indistinguishable by a polynomial-time adversary" defined over probability distributions
- Notice that $\approx$ is reflexive, symmetric, and transitive when applied a polynomial number of times

$$(pk, \mathsf{Enc}^{hy}_{pk}(m_0)) = (pk, c, \mathsf{Enc}'_k(m_0)) \quad \underset{\text{we want this}}{\overset{\approx}{\longleftrightarrow}} \quad (pk, \mathsf{Enc}^{hy}_{pk}(m_1))$$

CPA-Security of Hybrid encryption

Theorem: *If Π is a CPA-secure KEM and Π' is an EAV-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CPA-secure public-key encryption scheme.*

High-level idea of the proof:

- Let k^* be a key chosen u.a.r. from $\{0,1\}^{\ell(n)}$
- Let $\approx$ denote the relation "being indistinguishable by a polynomial-time adversary" defined over probability distributions
- Notice that $\approx$ is reflexive, symmetric, and transitive when applied a polynomial number of times

$$(pk, \mathsf{Enc}^{hy}_{pk}(m_0)) \;\; = (pk, c, \mathsf{Enc}'_k(m_0)) \;\overset{\approx}{\underset{\text{we want this}}{\longleftrightarrow}}\; (pk, c, \mathsf{Enc}'_k(m_1)) = \;\; (pk, \mathsf{Enc}^{hy}_{pk}(m_1))$$

CPA-Security of Hybrid encryption

Theorem: *If Π is a CPA-secure KEM and Π' is an EAV-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CPA-secure public-key encryption scheme.*

High-level idea of the proof:

- Let k^* be a key chosen u.a.r. from $\{0,1\}^{\ell(n)}$
- Let $\approx$ denote the relation "being indistinguishable by a polynomial-time adversary" defined over probability distributions
- Notice that $\approx$ is reflexive, symmetric, and transitive when applied a polynomial number of times

$$(pk, \mathsf{Enc}^{hy}_{pk}(m_0)) = (pk, c, \mathsf{Enc}'_k(m_0)) \overset{\approx}{\longleftrightarrow} (pk, c, \mathsf{Enc}'_k(m_1)) = (pk, \mathsf{Enc}^{hy}_{pk}(m_1))$$

we want this

From CPA-security of the KEM Π

(can't tell k and k^* apart)

$$(pk, c, \mathsf{Enc}'_k(m_0)) \approx (pk, c, \mathsf{Enc}'_{\boxed{k^*}}(m_0))$$

CPA-Security of Hybrid encryption

Theorem: *If Π is a CPA-secure KEM and Π' is an EAV-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CPA-secure public-key encryption scheme.*

High-level idea of the proof:

- Let k^* be a key chosen u.a.r. from $\{0,1\}^{\ell(n)}$
- Let $\approx$ denote the relation "being indistinguishable by a polynomial-time adversary" defined over probability distributions
- Notice that $\approx$ is reflexive, symmetric, and transitive when applied a polynomial number of times

$$
\begin{array}{ccccc}
(pk, \mathsf{Enc}^{hy}_{pk}(m_0)) = (pk, c, \mathsf{Enc}'_k(m_0)) & \xleftrightarrow[\text{we want this}]{\approx} & (pk, c, \mathsf{Enc}'_k(m_1)) = (pk, \mathsf{Enc}^{hy}_{pk}(m_1)) \\
\updownarrow \approx & & \updownarrow \approx \\
(pk, c, \mathsf{Enc}'_{k^*}(m_0)) & & (pk, c, \mathsf{Enc}'_{k^*}(m_1))
\end{array}
$$

From CPA-security of the KEM Π

(can't tell k and k^* apart)

CPA-Security of Hybrid encryption

Theorem: *If Π is a CPA-secure KEM and Π' is an EAV-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CPA-secure public-key encryption scheme.*

High-level idea of the proof:

- Let k^* be a key chosen u.a.r. from $\{0,1\}^{\ell(n)}$
- Let $\approx$ denote the relation "being indistinguishable by a polynomial-time adversary" defined over probability distributions
- Notice that $\approx$ is reflexive, symmetric, and transitive when applied a polynomial number of times

$$
\begin{tikzcd}
(pk, \mathsf{Enc}^{hy}_{pk}(m_0)) = (pk, c, \mathsf{Enc}'_k(m_0)) \arrow[r, leftrightarrow, dashed, "\approx", "\text{we want this}"'] \arrow[d, leftrightarrow, "\approx"] & (pk, c, \mathsf{Enc}'_k(m_1)) = (pk, \mathsf{Enc}^{hy}_{pk}(m_1)) \arrow[d, leftrightarrow, "\approx"] \\
(pk, c, \mathsf{Enc}'_{k^*}(m_0)) \arrow[r, leftrightarrow, "\approx"] & (pk, c, \mathsf{Enc}'_{k^*}(m_1))
\end{tikzcd}
$$

From CPA-security of the KEM Π

(can't tell k and k^* apart)

From EAV-security of the DEM Π'

Security Against Chosen Ciphertext Attacks

Let $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM and let $\mathcal{A}$ be an algorithm. We denote the following experiment by $\mathsf{KEM}^{\mathsf{cca}}_{\mathcal{A},\Pi}$

- A key pair $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ is generated
- $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$
- A uniform random bit $b \in \{0, 1\}$ is generated
- If $b = 0$, we set $\widehat{k} = k$. Otherwise $(b = 1)$ we choose $\widehat{k}$ u.a.r. from $\{0, 1\}^{\ell(n)}$
- The values of pk, c, and $\widehat{k}$ are sent to $\mathcal{A}$
- $\mathcal{A}$ interacts with an oracle providing access to $\mathsf{Decaps}_{sk}(\cdot)$, but it cannot query the oracle with c
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ about b
- The *outcome of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

Security Against Chosen Ciphertext Attacks

Let $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM and let $\mathcal{A}$ be an algorithm. We denote the following experiment by $\mathsf{KEM}^{\mathsf{cca}}_{\mathcal{A},\Pi}$

- A key pair $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ is generated
- $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$
- A uniform random bit $b \in \{0, 1\}$ is generated
- If $b = 0$, we set $\widehat{k} = k$. Otherwise $(b = 1)$ we choose $\widehat{k}$ u.a.r. from $\{0, 1\}^{\ell(n)}$
- The values of pk, c, and $\widehat{k}$ are sent to $\mathcal{A}$
- $\mathcal{A}$ interacts with an oracle providing access to $\mathsf{Decaps}_{sk}(\cdot)$, but it cannot query the oracle with c
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ about b
- The *outcome of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

Definition: *A key-encapsulation mechanism Π is* **CCA-secure** *if, for every probabilistic polynomial-time adversary $\mathcal{A}$, there is a negligible function ε such that:*

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

CCA-Security of Hybrid encryption

We can combine a CCA-secure KEM and a CCA-secure DEM to obtain a CCA-secure public-key encryption scheme:

> **Theorem**: *If Π is a CCA-secure KEM and Π' is an CCA-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CCA-secure public-key encryption scheme.*

CCA-Security of Hybrid encryption

We can combine a CCA-secure KEM and a CCA-secure DEM to obtain a CCA-secure public-key encryption scheme:

> **Theorem**: *If Π is a CCA-secure KEM and Π' is an CCA-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CCA-secure public-key encryption scheme.*

Why do we need both Π and Π' to be CCA-secure?

CCA-Security of Hybrid encryption

We can combine a CCA-secure KEM and a CCA-secure DEM to obtain a CCA-secure public-key encryption scheme:

> **Theorem**: *If Π is a CCA-secure KEM and Π' is an CCA-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CCA-secure public-key encryption scheme.*

Why do we need both Π and Π' to be CCA-secure?

- If Π is not CCA-secure then the adversary can attack the encapsulation mechanism and gain information about/recover the DEM key.

CCA-Security of Hybrid encryption

We can combine a CCA-secure KEM and a CCA-secure DEM to obtain a CCA-secure public-key encryption scheme:

> **Theorem**: *If Π is a CCA-secure KEM and Π' is an CCA-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CCA-secure public-key encryption scheme.*

Why do we need both Π and Π' to be CCA-secure?

- If Π is not CCA-secure then the adversary can attack the encapsulation mechanism and gain information about/recover the DEM key.
- If Π' is not CCA-secure then the adversary can gain information on the encrypted message by attacking the DEM

CCA-Security of Hybrid encryption

We can combine a CCA-secure KEM and a CCA-secure DEM to obtain a CCA-secure public-key encryption scheme:

> **Theorem**: *If Π is a CCA-secure KEM and Π' is an CCA-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CCA-secure public-key encryption scheme.*

Why do we need both Π and Π' to be CCA-secure?

- If Π is not CCA-secure then the adversary can attack the encapsulation mechanism and gain information about/recover the DEM key.
- If Π' is not CCA-secure then the adversary can gain information on the encrypted message by attacking the DEM

Consider the hybrid encryption using some CCA-secure KEM and the pseudo-OTP DEM.

The ciphertext is: $\langle c, c' \rangle = \langle c, G(k) \oplus m \rangle$, where G is a PRG.

CCA-Security of Hybrid encryption

We can combine a CCA-secure KEM and a CCA-secure DEM to obtain a CCA-secure public-key encryption scheme:

> **Theorem**: *If Π is a CCA-secure KEM and Π' is an CCA-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CCA-secure public-key encryption scheme.*

Why do we need both Π and Π' to be CCA-secure?

- If Π is not CCA-secure then the adversary can attack the encapsulation mechanism and gain information about/recover the DEM key.
- If Π' is not CCA-secure then the adversary can gain information on the encrypted message by attacking the DEM

Consider the hybrid encryption using some CCA-secure KEM and the pseudo-OTP DEM.

The ciphertext is: $\langle c, c' \rangle = \langle c, G(k) \oplus m \rangle$, where G is a PRG. **The DEM is malleable!**

CCA-Security of Hybrid encryption

We can combine a CCA-secure KEM and a CCA-secure DEM to obtain a CCA-secure public-key encryption scheme:

Theorem: *If Π is a CCA-secure KEM and Π' is an CCA-secure private-key encryption scheme then Π^{hy} (as previously defined) is a CCA-secure public-key encryption scheme.*

Why do we need both Π and Π' to be CCA-secure?

- If Π is not CCA-secure then the adversary can attack the encapsulation mechanism and gain information about/recover the DEM key.
- If Π' is not CCA-secure then the adversary can gain information on the encrypted message by attacking the DEM

Consider the hybrid encryption using some CCA-secure KEM and the pseudo-OTP DEM.

The ciphertext is: $\langle c, c' \rangle = \langle c, G(k) \oplus m \rangle$, where G is a PRG. **The DEM is malleable!**

The adversary can compute $c'' = c' \oplus \underbrace{00\ldots0}_{|c'|-1 \text{ times}} 1$, query the decryption oracle with c'' to obtain

$m'' = \text{Dec}_{sk}(\langle c, c'' \rangle)$, and recover $m = m'' \oplus 00\ldots01$

El Gamal Encryption

El Gamal Encryption

The Diffie-Hellman key-exchange protocol allows Alice and Bob to agree on a secret shared group element $k \in G$ (for some group G)

El Gamal Encryption

The Diffie-Hellman key-exchange protocol allows Alice and Bob to agree on a secret shared group element $k \in G$ (for some group G)

The element k is indistinguishable from a random element from G to any polynomial-time adversary

El Gamal Encryption

The Diffie-Hellman key-exchange protocol allows Alice and Bob to agree on a secret shared group element $k \in G$ (for some group G)

The element k is indistinguishable from a random element from G to any polynomial-time adversary

We can think of the shared element k as a "private key" that can be used by Bob to encrypt a message $m \in G$

El Gamal Encryption

The Diffie-Hellman key-exchange protocol allows Alice and Bob to agree on a secret shared group element $k \in G$ (for some group G)

The element k is indistinguishable from a random element from G to any polynomial-time adversary

We can think of the shared element k as a "private key" that can be used by Bob to encrypt a message $m \in G$

The message is also a group element!

El Gamal Encryption

The Diffie-Hellman key-exchange protocol allows Alice and Bob to agree on a secret shared group element $k \in G$ (for some group G)

The element k is indistinguishable from a random element from G to any polynomial-time adversary

We can think of the shared element k as a "private key" that can be used by Bob to encrypt a message $m \in G$

The message is also a group element!

In the following Bob will be sending a message to Alice!

El Gamal Encryption

The Diffie-Hellman key-exchange protocol allows Alice and Bob to agree on a secret shared group element $k \in G$ (for some group G)

The element k is indistinguishable from a random element from G to any polynomial-time adversary

We can think of the shared element k as a "private key" that can be used by Bob to encrypt a message $m \in G$

The message is also a group element!

In the following Bob will be sending a message to Alice!

To encrypt m, Bob simply computes $c = m \cdot k$ (the product denotes the group operation)

El Gamal Encryption

The Diffie-Hellman key-exchange protocol allows Alice and Bob to agree on a secret shared group element $k \in G$ (for some group G)

The element k is indistinguishable from a random element from G to any polynomial-time adversary

We can think of the shared element k as a "private key" that can be used by Bob to encrypt a message $m \in G$

The message is also a group element!

In the following Bob will be sending a message to Alice!

To encrypt m, Bob simply computes $c = m \cdot k$ (the product denotes the group operation)

Lemma: Let G be a group, let $m \in G$, and let k be a group element chosen u.a.r. Define $c = k \cdot m$. For any $\widehat{c} \in G$ we have: $\Pr[c = \widehat{c}] = \frac{1}{|G|}$.

El Gamal Encryption

The Diffie-Hellman key-exchange protocol allows Alice and Bob to agree on a secret shared group element $k \in G$ (for some group G)

The element k is indistinguishable from a random element from G to any polynomial-time adversary

We can think of the shared element k as a "private key" that can be used by Bob to encrypt a message $m \in G$

The message is also a group element!

In the following Bob will be sending a message to Alice!

To encrypt m, Bob simply computes $c = m \cdot k$ (the product denotes the group operation)

Lemma: Let G be a group, let $m \in G$, and let k be a group element chosen u.a.r. Define $c = k \cdot m$. For any $\widehat{c} \in G$ we have: $\Pr[c = \widehat{c}] = \frac{1}{|G|}$.

Proof:

$$\Pr\left[c = \widehat{c}\right] = \Pr\left[k \cdot m = \widehat{c}\right]$$

El Gamal Encryption

The Diffie-Hellman key-exchange protocol allows Alice and Bob to agree on a secret shared group element $k \in G$ (for some group G)

The element k is indistinguishable from a random element from G to any polynomial-time adversary

We can think of the shared element k as a "private key" that can be used by Bob to encrypt a message $m \in G$

The message is also a group element!

In the following Bob will be sending a message to Alice!

To encrypt m, Bob simply computes $c = m \cdot k$ (the product denotes the group operation)

Lemma: Let G be a group, let $m \in G$, and let k be a group element chosen u.a.r. Define $c = k \cdot m$. For any $\widehat{c} \in G$ we have: $\Pr[c = \widehat{c}] = \frac{1}{|G|}$.

Proof:

$$\Pr\left[c = \widehat{c}\right] = \Pr\left[k \cdot m = \widehat{c}\right] = \Pr\left[k \cdot m \cdot m^{-1} = \widehat{c} \cdot m^{-1}\right]$$

El Gamal Encryption

The Diffie-Hellman key-exchange protocol allows Alice and Bob to agree on a secret shared group element $k \in G$ (for some group G)

The element k is indistinguishable from a random element from G to any polynomial-time adversary

We can think of the shared element k as a "private key" that can be used by Bob to encrypt a message $m \in G$

The message is also a group element!

In the following Bob will be sending a message to Alice!

To encrypt m, Bob simply computes $c = m \cdot k$ (the product denotes the group operation)

Lemma: Let G be a group, let $m \in G$, and let k be a group element chosen u.a.r. Define $c = k \cdot m$. For any $\widehat{c} \in G$ we have: $\Pr[c = \widehat{c}] = \frac{1}{|G|}$.

Proof:

$$\Pr\left[c = \widehat{c}\right] = \Pr\left[k \cdot m = \widehat{c}\right] = \Pr\left[k \cdot m \cdot m^{-1} = \widehat{c} \cdot m^{-1}\right] = \Pr\left[k = \widehat{c} \cdot m^{-1}\right]$$

El Gamal Encryption

The Diffie-Hellman key-exchange protocol allows Alice and Bob to agree on a secret shared group element $k \in G$ (for some group G)

The element k is indistinguishable from a random element from G to any polynomial-time adversary

We can think of the shared element k as a "private key" that can be used by Bob to encrypt a message $m \in G$

The message is also a group element!

In the following Bob will be sending a message to Alice!

To encrypt m, Bob simply computes $c = m \cdot k$ (the product denotes the group operation)

Lemma: Let G be a group, let $m \in G$, and let k be a group element chosen u.a.r. Define $c = k \cdot m$. For any $\widehat{c} \in G$ we have: $\Pr[c = \widehat{c}] = \frac{1}{|G|}$.

Proof:

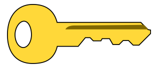
$$\Pr\left[c = \widehat{c}\right] = \Pr\left[k \cdot m = \widehat{c}\right] = \Pr\left[k \cdot m \cdot m^{-1} = \widehat{c} \cdot m^{-1}\right] = \Pr\left[k = \widehat{c} \cdot m^{-1}\right] = \frac{1}{|G|} \qquad \square$$

Diffie-Hellman Key Exchange $\rightarrow$ El Gamal Encryption

- Alice chooses a group G of order q and a generator $g \in G$.
- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send (G, q, g, h_A) to Bob

$\longrightarrow$

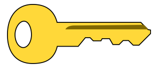
- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Compute $k = h_A^y = (g^x)^y = g^{xy}$



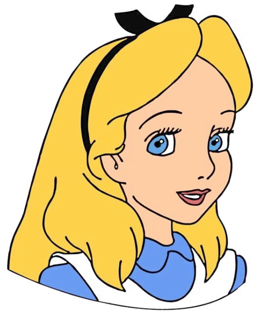
- Send h_B to Alice

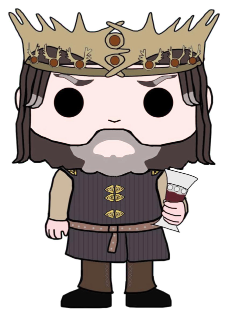
$\longleftarrow$

- Compute $k = h_B^x = (g^y)^x = g^{xy}$



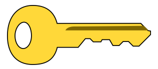
Diffie-Hellman Key Exchange $\to$ El Gamal Encryption

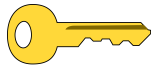




Insecure Channel

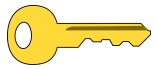
- Alice chooses a group G of order q and a generator $g \in G$.
- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send (G, q, g, h_A) to Bob

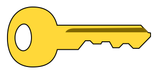
- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Compute $k = h_A^y = (g^x)^y = g^{xy}$ 
- Compute $c = k \cdot m$
- Send h_B to Alice

- Compute $k = h_B^x = (g^y)^x = g^{xy}$ 

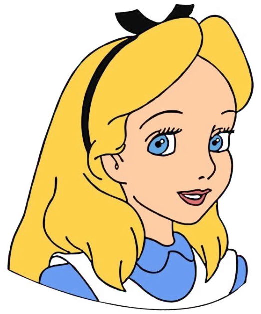
Diffie-Hellman Key Exchange $\rightarrow$ El Gamal Encryption

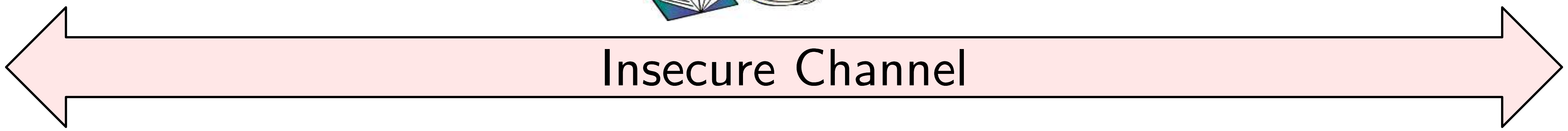
- Alice chooses a group G of order q and a generator $g \in G$.
- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send (G, q, g, h_A) to Bob

- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Compute $k = h_A^y = (g^x)^y = g^{xy}$ 
- Compute $c = k \cdot m$
- Send $\langle h_B, c \rangle$ to Alice

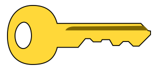
- Compute $k = h_B^x = (g^y)^x = g^{xy}$ 

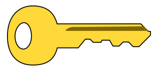
Diffie-Hellman Key Exchange $\to$ El Gamal Encryption





- Alice chooses a group G of order q and a generator $g \in G$.
- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send (G, q, g, h_A) to Bob

- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Compute $k = h_A^y = (g^x)^y = g^{xy}$ 
- Compute $c = k \cdot m$
- Send $\langle h_B, c \rangle$ to Alice

- Compute $k = h_B^x = (g^y)^x = g^{xy}$ 
- Compute $m = k^{-1} \cdot c$

Diffie-Hellman Key Exchange $\rightarrow$ El Gamal Encryption

- Alice chooses a group G of order q and a generator $g \in G$.
- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send (G, q, g, h_A) to Bob

Alice's public key pk

- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Compute $k = h_A^y = (g^x)^y = g^{xy}$
- Compute $c = k \cdot m$
- Send $\langle h_B, c \rangle$ to Alice

- Compute $k = h_B^x = (g^y)^x = g^{xy}$
- Compute $m = k^{-1} \cdot c$

Diffie-Hellman Key Exchange $\rightarrow$ El Gamal Encryption

- Alice chooses a group G of order q and a generator $g \in G$.
- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send (G, q, g, h_A) to Bob

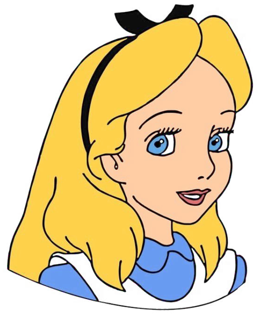
Alice's secret key sk

Alice's public key pk

- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Compute $k = h_A^y = (g^x)^y = g^{xy}$
- Compute $c = k \cdot m$
- Send $\langle h_B, c \rangle$ to Alice

- Compute $k = h_B^x = (g^y)^x = g^{xy}$
- Compute $m = k^{-1} \cdot c$

Diffie-Hellman Key Exchange → El Gamal Encryption



Insecure Channel

Key generation

- Alice chooses a group G of order q and a generator $g \in G$.
- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send (G, q, g, h_A) to Bob

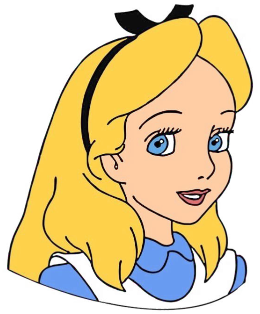
Alice's secret key sk

Alice's public key pk

- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Compute $k = h_A^y = (g^x)^y = g^{xy}$
- Compute $c = k \cdot m$
- Send $\langle h_B, c \rangle$ to Alice

- Compute $k = h_B^x = (g^y)^x = g^{xy}$
- Compute $m = k^{-1} \cdot c$

Diffie-Hellman Key Exchange $\to$ El Gamal Encryption



Insecure Channel

Key generation

- Alice chooses a group G of order q and a generator $g \in G$.
- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send (G, q, g, h_A) to Bob

Alice's secret key sk

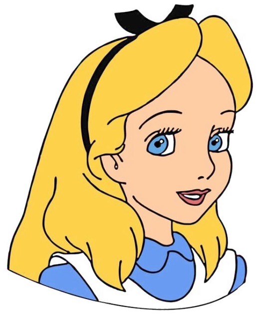
Alice's public key pk

Encryption

- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Compute $k = h_A^y = (g^x)^y = g^{xy}$
- Compute $c = k \cdot m$
- Send $\langle h_B, c \rangle$ to Alice

- Compute $k = h_B^x = (g^y)^x = g^{xy}$
- Compute $m = k^{-1} \cdot c$

Diffie-Hellman Key Exchange $\to$ El Gamal Encryption



Insecure Channel

Key generation

- Alice chooses a group G of order q and a generator $g \in G$.
- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send (G, q, g, h_A) to Bob

Alice's secret key sk

Alice's public key pk

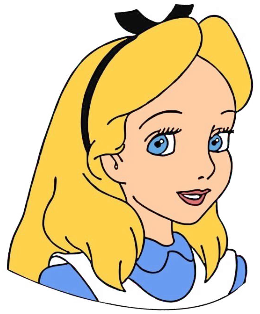
Encryption

- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Compute $k = h_A^y = (g^x)^y = g^{xy}$
- Compute $c = k \cdot m$
- Send $\langle h_B, c \rangle$ to Alice

Decryption

- Compute $k = h_B^x = (g^y)^x = g^{xy}$
- Compute $m = k^{-1} \cdot c$

Diffie-Hellman Key Exchange $\rightarrow$ El Gamal Encryption



Insecure Channel

Key generation

- Alice chooses a group G of order q and a generator $g \in G$.
- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send (G, q, g, h_A) to Bob

Alice's secret key sk

Alice's public key pk

We (essentially) have a public-key encryption scheme!

Encryption

- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Compute $k = h_A^y = (g^x)^y = g^{xy}$
- Compute $c = k \cdot m$
- Send $\langle h_B, c \rangle$ to Alice

Decryption

- Compute $k = h_B^x = (g^y)^x = g^{xy}$
- Compute $m = k^{-1} \cdot c$

El Gamal Encryption (more formally)

Gen(1^n):

- Run $\mathcal{G}(1^n)$, where $\mathcal{G}$ is a group generation algorithm, to obtain (G, q, g) where G is a group of order q and $g \in G$ is a generator
- Choose a uniform x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h = g^x$
- Output (pk, sk) where $pk = (G, q, g, h)$ and $sk = (G, q, g, x)$.

The message space $\mathcal{M}_{pk}$ is G.

Enc$_{pk}(m)$:

- Here $pk = (G, q, g, h)$ and $m \in G$
- Choose a uniform y u.a.r. from $\{0, \ldots, q-1\}$
- Output the ciphertext $\langle g^y, h^y \cdot m \rangle$

Dec$_{sk}(c)$:

- Here $sk = (G, q, g, x)$ and $c = \langle c_1, c_2 \rangle$
- Output the plaintext $(c_1^x)^{-1} \cdot c_2$

El Gamal Encryption (more formally)

$\mathsf{Gen}(1^n)$:

- Run $\mathcal{G}(1^n)$, where $\mathcal{G}$ is a group generation algorithm, to obtain (G, q, g) where G is a group of order q and $g \in G$ is a generator
- Choose a uniform x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h = g^x$
- Output (pk, sk) where $pk = (G, q, g, h)$ and $sk = (G, q, g, x)$.

The message space $\mathcal{M}_{pk}$ is G.

$\mathsf{Enc}_{pk}(m)$:

- Here $pk = (G, q, g, h)$ and $m \in G$
- Choose a uniform y u.a.r. from $\{0, \ldots, q-1\}$
- Output the ciphertext $\langle g^y, h^y \cdot m \rangle$

$\mathsf{Dec}_{sk}(c)$:

- Here $sk = (G, q, g, x)$ and $c = \langle c_1, c_2 \rangle$
- Output the plaintext $(c_1^x)^{-1} \cdot c_2$

In practice the group G (and its order q) is fixed in advance along with a generator $g \in G$.

(just like in the Diffie-Hellman key exchange).

Security of El Gamal Encryption

> If the DDH problem is hard relative to G, then the El Gamal encryption scheme Π is CPA-secure.

Security of El Gamal Encryption

If the DDH problem is hard relative to G, then the El Gamal encryption scheme Π is CPA-secure.

Proof:

We show that a polynomial-time adversary that wins the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment can be used to solve the DDH problem with non-negligible advantage (in polynomial time).

Suppose towards a contradiction that there is some polynomial-time algorithm $\mathcal{A}$ such that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.

Security of El Gamal Encryption

If the DDH problem is hard relative to G, then the El Gamal encryption scheme Π is CPA-secure.

Proof:

We show that a polynomial-time adversary that wins the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment can be used to solve the DDH problem with non-negligible advantage (in polynomial time).

Suppose towards a contradiction that there is some polynomial-time algorithm $\mathcal{A}$ such that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.

Let $\widetilde{\Pi}$ be a modified version of Π in which encryption is done by selecting $y, z \in \{0, 1, \ldots, q-1\}$ u.a.r., and returning $\langle g^y, g^z \rangle$.

Security of El Gamal Encryption

> If the DDH problem is hard relative to G, then the El Gamal encryption scheme Π is CPA-secure.

Proof:

We show that a polynomial-time adversary that wins the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment can be used to solve the DDH problem with non-negligible advantage (in polynomial time).

Suppose towards a contradiction that there is some polynomial-time algorithm $\mathcal{A}$ such that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.

Let $\widetilde{\Pi}$ be a modified version of Π in which encryption is done by selecting $y, z \in \{0, 1, \ldots, q-1\}$ u.a.r., and returning $\langle g^y, g^z \rangle$.

- Notice that $\widetilde{\Pi}$ is **not** a correct public-key encryption scheme (decryption breaks).

Security of El Gamal Encryption

If the DDH problem is hard relative to G, then the El Gamal encryption scheme Π is CPA-secure.

Proof:

We show that a polynomial-time adversary that wins the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment can be used to solve the DDH problem with non-negligible advantage (in polynomial time).

Suppose towards a contradiction that there is some polynomial-time algorithm $\mathcal{A}$ such that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.

Let $\widetilde{\Pi}$ be a modified version of Π in which encryption is done by selecting $y, z \in \{0, 1, \ldots, q-1\}$ u.a.r., and returning $\langle g^y, g^z \rangle$.

- Notice that $\widetilde{\Pi}$ is **not** a correct public-key encryption scheme (decryption breaks).
- We can still talk about the experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n)$, since we never use Dec

Security of El Gamal Encryption

If the DDH problem is hard relative to G, then the El Gamal encryption scheme Π is CPA-secure.

Proof:

We show that a polynomial-time adversary that wins the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment can be used to solve the DDH problem with non-negligible advantage (in polynomial time).

Suppose towards a contradiction that there is some polynomial-time algorithm $\mathcal{A}$ such that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.

Let $\widetilde{\Pi}$ be a modified version of Π in which encryption is done by selecting $y, z \in \{0, 1, \ldots, q-1\}$ u.a.r., and returning $\langle g^y, g^z \rangle$.

- Notice that $\widetilde{\Pi}$ is **not** a correct public-key encryption scheme (decryption breaks).
- We can still talk about the experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n)$, since we never use Dec
- The ciphertext $\langle g^y, g^z \rangle$ is completely independent from m!

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the “public key” $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'
- If $b = b'$ output 1, otherwise output 0

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'
- If $b = b'$ output 1, otherwise output 0

If h is a random group element:

- By the previous lemma, $h \cdot m_b$ is distributed identically as picking a random $z \in \{0, 1, \ldots, q-1\}$ and computing g^z

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'
- If $b = b'$ output 1, otherwise output 0

If h is a random group element:

- By the previous lemma, $h \cdot m_b$ is distributed identically as picking a random $z \in \{0, 1, \dots, q-1\}$ and computing g^z
- Algorithm D is carrying out the $\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n)$ experiment!

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'
- If $b = b'$ output 1, otherwise output 0

If h is a random group element:

- By the previous lemma, $h \cdot m_b$ is distributed identically as picking a random $z \in \{0, 1, \ldots, q-1\}$ and computing g^z
- Algorithm D is carrying out the $\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n)$ experiment!
- $\Pr[D(G, q, g, g^x, g^y, g^z) = 1] = \Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'
- If $b = b'$ output 1, otherwise output 0

If $h = g^{xy}$:

- Algorithm D is carrying out the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment!

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'
- If $b = b'$ output 1, otherwise output 0

If $h = g^{xy}$:

- Algorithm D is carrying out the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment!
- $\Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] = \Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n) = 1]$

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'
- If $b = b'$ output 1, otherwise output 0

If $h = g^{xy}$:

- Algorithm D is carrying out the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment!
- $\Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] = \Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n) = 1] = \frac{1}{2} + \varepsilon(n)$

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'
- If $b = b'$ output 1, otherwise output 0

If $h = g^{xy}$:

- Algorithm D is carrying out the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment!
- $\Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] = \Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n) = 1] \ = \frac{1}{2} + \varepsilon(n)$

$$\Big| \Pr[D(G, q, g, g^x, g^y, g^z) = 1] - \Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] \Big|$$

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'
- If $b = b'$ output 1, otherwise output 0

If $h = g^{xy}$:

- Algorithm D is carrying out the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment!
- $\Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] = \Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n) = 1] \; = \frac{1}{2} + \varepsilon(n)$

$$\left| \Pr[D(G, q, g, g^x, g^y, g^z) = 1] - \Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] \right| \quad = \left| \tfrac{1}{2} - (\tfrac{1}{2} + \varepsilon(n)) \right|$$

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'
- If $b = b'$ output 1, otherwise output 0

If $h = g^{xy}$:

- Algorithm D is carrying out the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment!
- $\Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] = \Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n) = 1] \ = \frac{1}{2} + \varepsilon(n)$

$$\left| \Pr[D(G, q, g, g^x, g^y, g^z) = 1] - \Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] \right| \quad = \left| \frac{1}{2} - (\frac{1}{2} + \varepsilon(n)) \right| \ = \varepsilon(n)$$

Security of El Gamal Encryption (cont.)

It follows that $\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\widetilde{\Pi},\mathcal{A}}(n) = 1] = \frac{1}{2}$

Consider the following algorithm D for the DDH problem:

- Receive as input G, q, g, g^x, g^y, h
- Send the "public key" $pk = \langle G, q, g, g^x \rangle$ to $\mathcal{A}$ and receive two messages m_0, m_1
- Pick a uniform $b \in \{0, 1\}$
- Give the ciphertext $\langle g^y, h \cdot m_b \rangle$ to $\mathcal{A}$ and obtain a guess b'
- If $b = b'$ output 1, otherwise output 0

If $h = g^{xy}$:

- Algorithm D is carrying out the $\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n)$ experiment!
- $\Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] = \Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(n) = 1] \; = \frac{1}{2} + \varepsilon(n)$

not negligible!

$$\left| \Pr[D(G, q, g, g^x, g^y, g^z) = 1] - \Pr[D(G, q, g, g^x, g^y, g^{xy}) = 1] \right| \; = \left| \frac{1}{2} - \left(\frac{1}{2} + \varepsilon(n)\right) \right| \; = \boxed{\varepsilon(n)} \; \lightning \quad \square$$

Some Remarks

We just built a (fixed-length) public-key encryption scheme. However there are some drawbacks:

- The message space depends on the public-key (i.e., on the choice of the group G)

Some Remarks

We just built a (fixed-length) public-key encryption scheme. However there are some drawbacks:

- The message space depends on the public-key (i.e., on the choice of the group G)
- Even if we think of G as fixed, we can only encrypt group elements

Some Remarks

We just built a (fixed-length) public-key encryption scheme. However there are some drawbacks:

- The message space depends on the public-key (i.e., on the choice of the group G)
- Even if we think of G as fixed, we can only encrypt group elements
- We would like to encrypt arbitrary (fixed-length) binary strings

Some Remarks

We just built a (fixed-length) public-key encryption scheme. However there are some drawbacks:

- The message space depends on the public-key (i.e., on the choice of the group G)
- Even if we think of G as fixed, we can only encrypt group elements
- We would like to encrypt arbitrary (fixed-length) binary strings

One way around this issue, is that of agreeing on an encoding of strings into group elements

- Depending on the group this might not be easy to do (recall that we need the encoding to be computable and invertible in polynomial-time)

Some Remarks

We just built a (fixed-length) public-key encryption scheme. However there are some drawbacks:

- The message space depends on the public-key (i.e., on the choice of the group G)
- Even if we think of G as fixed, we can only encrypt group elements
- We would like to encrypt arbitrary (fixed-length) binary strings

One way around this issue, is that of agreeing on an encoding of strings into group elements

- Depending on the group this might not be easy to do (recall that we need the encoding to be computable and invertible in polynomial-time)

As an alternative, we can use our public-key encryption scheme as a KEM for Hybrid Encryption

El Gamal Key Encapsulation Mechanism

El Gamal as a KEM:

- Gen: as before
- Encaps: pick a random group element $\widetilde{k} \in G$ and encrypt it as $\langle g^y, h^y \cdot \widetilde{k} \rangle$. Return $(k, \langle g^y, h^y \cdot \widetilde{k} \rangle)$
- Decaps: given $\langle c_1, c_2 \rangle$ compute $\widetilde{k} = (c_1^x)^{-1} \cdot c_2$. Return $\widetilde{k}$.

El Gamal Key Encapsulation Mechanism

El Gamal as a KEM:

- Gen: as before
- Encaps: pick a random group element $\widetilde{k} \in G$ and encrypt it as $\langle g^y, h^y \cdot \widetilde{k} \rangle$. Return $(k, \langle g^y, h^y \cdot \widetilde{k} \rangle)$
- Decaps: given $\langle c_1, c_2 \rangle$ compute $\widetilde{k} = (c_1^x)^{-1} \cdot c_2$. Return $\widetilde{k}$.

Note: to get a binary-string key k from $\widetilde{k}$, we need to use a key-derivation function $H : G \to \{0,1\}^{\ell(n)}$.

El Gamal Key Encapsulation Mechanism

El Gamal as a KEM:

- Gen: as before
- Encaps: pick a random group element $\widetilde{k} \in G$ and encrypt it as $\langle g^y, h^y \cdot \widetilde{k} \rangle$. Return $(k, \langle g^y, h^y \cdot \widetilde{k} \rangle)$
- Decaps: given $\langle c_1, c_2 \rangle$ compute $\widetilde{k} = (c_1^x)^{-1} \cdot c_2$. Return $\widetilde{k}$.

Note: to get a binary-string key k from $\widetilde{k}$, we need to use a key-derivation function $H : G \to \{0,1\}^{\ell(n)}$.

Drawback: we are sharing **only a single** secret group element $\widetilde{k}$ but the ciphertext consists of **two** group elements, i.e., it is $\langle g^y, h^y \cdot \widetilde{k} \rangle$.

El Gamal Key Encapsulation Mechanism

El Gamal as a KEM:

- Gen: as before
- Encaps: pick a random group element $\widetilde{k} \in G$ and encrypt it as $\langle g^y, h^y \cdot \widetilde{k} \rangle$. Return $(k, \langle g^y, h^y \cdot \widetilde{k} \rangle)$
- Decaps: given $\langle c_1, c_2 \rangle$ compute $\widetilde{k} = (c_1^x)^{-1} \cdot c_2$. Return $\widetilde{k}$.

Note: to get a binary-string key k from $\widetilde{k}$, we need to use a key-derivation function $H : G \to \{0,1\}^{\ell(n)}$.

Drawback: we are sharing **only a single** secret group element $\widetilde{k}$ but the ciphertext consists of **two** group elements, i.e., it is $\langle g^y, h^y \cdot \widetilde{k} \rangle$.

Idea: From the security proof Diffie-Hellman we know that $c_1^x = g^{xy}$ is already indistinguishable from a random group element (to a polynomial-time adversary).

El Gamal Key Encapsulation Mechanism

El Gamal as a KEM:

- Gen: as before
- Encaps: pick a random group element $\widetilde{k} \in G$ and encrypt it as $\langle g^y, h^y \cdot \widetilde{k} \rangle$. Return $(k, \langle g^y, h^y \cdot \widetilde{k} \rangle)$
- Decaps: given $\langle c_1, c_2 \rangle$ compute $\widetilde{k} = (c_1^x)^{-1} \cdot c_2$. Return $\widetilde{k}$.

Note: to get a binary-string key k from $\widetilde{k}$, we need to use a key-derivation function $H : G \to \{0,1\}^{\ell(n)}$.

Drawback: we are sharing **only a single** secret group element $\widetilde{k}$ but the ciphertext consists of **two** group elements, i.e., it is $\langle g^y, h^y \cdot \widetilde{k} \rangle$.

Idea: From the security proof Diffie-Hellman we know that $c_1^x = g^{xy}$ is already indistinguishable from a random group element (to a polynomial-time adversary).

Use c_1^x as the key!

DDH-Based Key Encapsulation Mechanism

$\mathsf{Gen}(1^n)$:

- Run $\mathcal{G}(1^n)$, where $\mathcal{G}$ is a group generation algorithm, to obtain (G, q, g) where G is a group of order q and $g \in G$ is a generator
- Choose a uniform x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h = g^x$
- Pick some key derivation function $H : G \to \{0,1\}^{\ell(n)}$
- Output (pk, sk) where $pk = (G, q, g, h, H)$ and $sk = (G, q, g, x, H)$.

DDH-Based Key Encapsulation Mechanism

Gen(1^n):

- Run $\mathcal{G}(1^n)$, where $\mathcal{G}$ is a group generation algorithm, to obtain (G, q, g) where G is a group of order q and $g \in G$ is a generator
- Choose a uniform x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h = g^x$
- Pick some key derivation function $H : G \to \{0,1\}^{\ell(n)}$
- Output (pk, sk) where $pk = (G, q, g, h, H)$ and $sk = (G, q, g, x, H)$.

Encaps$_{pk}$(1^n):

- Here $pk = (G, q, g, h, H)$
- Choose a uniform y u.a.r. from $\{0, \ldots, q-1\}$
- Output the pair (c, k) with $c = g^y$ and $k = H(h^y) = H(g^{xy})$

DDH-Based Key Encapsulation Mechanism

$\mathsf{Gen}(1^n)$:

- Run $\mathcal{G}(1^n)$, where $\mathcal{G}$ is a group generation algorithm, to obtain (G, q, g) where G is a group of order q and $g \in G$ is a generator
- Choose a uniform x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h = g^x$
- Pick some key derivation function $H : G \to \{0,1\}^{\ell(n)}$
- Output (pk, sk) where $pk = (G, q, g, h, H)$ and $sk = (G, q, g, x, H)$.

$\mathsf{Encaps}_{pk}(1^n)$:

- Here $pk = (G, q, g, h, H)$
- Choose a uniform y u.a.r. from $\{0, \ldots, q-1\}$
- Output the pair (c, k) with $c = g^y$ and $k = H(h^y) = H(g^{xy})$

$\mathsf{Decaps}_{sk}(c)$:

- Here $sk = (G, q, g, x, H)$
- Output the key $H(c^x) = H(g^{xy})$

The Key Derivation Function

How do we choose $H : G \to \{0,1\}^{\ell(n)}$?

The Key Derivation Function

How do we choose $H : G \to \{0, 1\}^{\ell(n)}$?

- Pick H as a hash function

The Key Derivation Function

How do we choose $H : G \to \{0, 1\}^{\ell(n)}$?

- Pick H as a hash function
  - Secure in the random oracle model

The Key Derivation Function

How do we choose $H : G \to \{0,1\}^{\ell(n)}$?

- Pick H as a hash function
  - Secure in the random oracle model
  - Can also be proven secure under the CDH assumption

The Key Derivation Function

How do we choose $H : G \to \{0,1\}^{\ell(n)}$?

- Pick H as a hash function
    - Secure in the random oracle model
    - Can also be proven secure under the CDH assumption
- Pick H as a "regular" function: the number of elements of G that map to the same key $k \in \{0,1\}^{\ell(n)}$ must be roughly the same.

The Key Derivation Function

How do we choose $H : G \to \{0,1\}^{\ell(n)}$?

- Pick H as a hash function
  - Secure in the random oracle model
  - Can also be proven secure under the CDH assumption
- Pick H as a "regular" function: the number of elements of G that map to the same key $k \in \{0,1\}^{\ell(n)}$ must be roughly the same.

  Formally, we need:

$$\sum_{k \in \{0,1\}^{\ell(n)}} \left| \Pr[H(g) = k] - 2^{-\ell(n)} \right| \leq \varepsilon(n),$$

  where $\varepsilon(n)$ is a negligible function and the probability is taken over the uniform choice of $g \in G$.

The Key Derivation Function

How do we choose $H : G \to \{0,1\}^{\ell(n)}$?

- Pick H as a hash function
  - Secure in the random oracle model
  - Can also be proven secure under the CDH assumption
- Pick H as a "regular" function: the number of elements of G that map to the same key $k \in \{0,1\}^{\ell(n)}$ must be roughly the same.

  Formally, we need:

$$\sum_{k \in \{0,1\}^{\ell(n)}} \left| \Pr[H(g) = k] - 2^{-\ell(n)} \right| \leq \varepsilon(n),$$

  where $\varepsilon(n)$ is a negligible function and the probability is taken over the uniform choice of $g \in G$.

If the DDH problem is hard relative to G, and H is chosen as above, then the DDH-based KEM is CPA-secure.

(Plain) RSA Encryption

Reminder: e-th roots

Let $N = pq$ where p and q are distinct odd primes

The order of Z_N^* is $\phi(N) = (p-1) \cdot (q-1)$

- Trivial to compute if we know p and q
- "Hard" to compute if we know N but not p and q (can be shown to be equivalent to factoring N)

Reminder: e-th roots

Let $N = pq$ where p and q are distinct odd primes

The order of Z_N^* is $\phi(N) = (p-1) \cdot (q-1)$

- Trivial to compute if we know p and q
- "Hard" to compute if we know N but not p and q (can be shown to be equivalent to factoring N)

Pick $e \in \mathbb{Z}_N^*$ such that $\gcd(e, \phi(N)) = 1$.

- $f_e(x) = x^e$ is a permutation of $\mathbb{Z}_N^*$
- Let d be the inverse of e modulo $\phi(N)$. Then $f_d(x) = x^d$ is the inverse of f_e.
- $(x^e)^d = (x^d)^e = x$

Reminder: e-th roots

Let $N = pq$ where p and q are distinct odd primes

The order of Z_N^* is $\phi(N) = (p-1) \cdot (q-1)$

- Trivial to compute if we know p and q
- "Hard" to compute if we know N but not p and q (can be shown to be equivalent to factoring N)

Pick $e \in \mathbb{Z}_N^*$ such that $\gcd(e, \phi(N)) = 1$.

- $f_e(x) = x^e$ is a permutation of $\mathbb{Z}_N^*$
- Let d be the inverse of e modulo $\phi(N)$. Then $f_d(x) = x^d$ is the inverse of f_e.
- $(x^e)^d = (x^d)^e = x$

Since $(x^e)^d = x$ we can think of x^d as the e-th root of x

- We define $x^{1/e} = x^d$

Reminder: the RSA problem

Let GenRSA be a polynomial-time algorithm that, on input 1^n , outputs a triple (N, e, d) where:

- $N = pq$, and p and q are n-bit primes
- $ed = 1 \pmod{\phi(N)}$

The algorithm is allowed to fail with negligible probability.

Reminder: the RSA problem

Let GenRSA be a polynomial-time algorithm that, on input 1^n , outputs a triple (N, e, d) where:

- $N = pq$, and p and q are n-bit primes
- $ed = 1 \pmod{\phi(N)}$

The algorithm is allowed to fail with negligible probability.

For an algorithm $\mathcal{A}$, define $\text{RSA-inv}_{\mathcal{A},\text{GenRSA}}(n)$ as:

- Run $\text{GenRSA}(1^n)$ to obtain (N, e, d).
- Choose $y \in \mathbb{Z}_N^*$ u.a.r.
- Send N, e and y to $\mathcal{A}$
- $\mathcal{A}$ outputs $x \in \mathbb{Z}_N^*$
- The outcome of the experiment is 1 if $x^e = y$. Otherwise the outcome is 0.

Reminder: the RSA assumption

Definition: The RSA problem is hard relative to GenRSA if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there exists a negligible function ε such that

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

The RSA assumption: there exists a GenRSA algorithm relative to which the RSA problem is hard.

RSA-Based Public Key Encryption

We can define a public-key encryption scheme (for short messages) based on the RSA assumption

Gen(1^n):

- $(N, e, d) \leftarrow \mathsf{GenRSA}(1^n)$
- Return (pk, sk) where $pk = \langle N, e \rangle$ and $sk = \langle N, d \rangle$

The message space $\mathcal{M}_{pk}$ is Z_N^*.

RSA-Based Public Key Encryption

We can define a public-key encryption scheme (for short messages) based on the RSA assumption

Gen(1^n):

- $(N, e, d) \leftarrow \mathsf{GenRSA}(1^n)$
- Return (pk, sk) where $pk = \langle N, e \rangle$ and $sk = \langle N, d \rangle$

The message space $\mathcal{M}_{pk}$ is Z_N^*.

Enc$_{pk}(m)$:

- Here $pk = \langle N, e \rangle$ and $m \in Z_N^*$
- $c \leftarrow m^e$ (the operation is in the group Z_N^*, under multiplication modulo N)
- Return c

RSA-Based Public Key Encryption

We can define a public-key encryption scheme (for short messages) based on the RSA assumption

$\mathsf{Gen}(1^n)$:

- $(N, e, d) \leftarrow \mathsf{GenRSA}(1^n)$
- Return (pk, sk) where $pk = \langle N, e \rangle$ and $sk = \langle N, d \rangle$

The message space $\mathcal{M}_{pk}$ is Z_N^*.

$\mathsf{Enc}_{pk}(m)$:

- Here $pk = \langle N, e \rangle$ and $m \in Z_N^*$
- $c \leftarrow m^e$ (the operation is in the group Z_N^*, under multiplication modulo N)
- Return c

$\mathsf{Dec}_{sk}(c)$:

- Here $sk = \langle N, d \rangle$ and $c \in Z_N^*$
- $m \leftarrow c^d$ (the operation is in the group Z_N^*, under multiplication modulo N)
- Return m

RSA-Based Public Key Encryption

We can define a public-key encryption scheme (for short messages) based on the RSA assumption

Gen(1^n):

- $(N, e, d) \leftarrow$ GenRSA(1^n)
- Return (pk, sk) where $pk = \langle N, e \rangle$ and $sk = \langle N, d \rangle$

The message space $\mathcal{M}_{pk}$ is Z_N^*.

Enc$_{pk}(m)$:

- Here $pk = \langle N, e \rangle$ and $m \in Z_N^*$
- $c \leftarrow m^e$ (the operation is in the group Z_N^*, under multiplication modulo N)
- Return c

Plain RSA

Dec$_{sk}(c)$:

- Here $sk = \langle N, d \rangle$ and $c \in Z_N^*$
- $m \leftarrow c^d$ (the operation is in the group Z_N^*, under multiplication modulo N)
- Return m

RSA-Based Public Key Encryption: Example

Say that we run $\texttt{GenRSA}(1^5)$ and it returns $(N, e, d) = (391, 3, 235)$

- We are going to work in the group $\mathbb{Z}^*_{391}$
- The public key pk is $(391, 3)$
- The secret key sk is $(391, 235)$

To encrypt $m = 158 \in \mathbb{Z}^*_{391}$:

RSA-Based Public Key Encryption: Example

Say that we run $\texttt{GenRSA}(1^5)$ and it returns $(N, e, d) = (391, 3, 235)$

- We are going to work in the group $\mathbb{Z}^*_{391}$
- The public key pk is $(391, 3)$
- The secret key sk is $(391, 235)$

To encrypt $m = 158 \in \mathbb{Z}^*_{391}$:

- Compute $c = 158^3 \bmod 391$

RSA-Based Public Key Encryption: Example

Say that we run $\texttt{GenRSA}(1^5)$ and it returns $(N, e, d) = (391, 3, 235)$

- We are going to work in the group $\mathbb{Z}^*_{391}$
- The public key pk is $(391, 3)$
- The secret key sk is $(391, 235)$

To encrypt $m = 158 \in \mathbb{Z}^*_{391}$:

- Compute $c = 158^3 \bmod 391 = (158^2 \bmod 391) \cdot 158 \bmod 391 = 331 \cdot 158 \bmod 391 = 295$

RSA-Based Public Key Encryption: Example

Say that we run $\texttt{GenRSA}(1^5)$ and it returns $(N, e, d) = (391, 3, 235)$

- We are going to work in the group $\mathbb{Z}^*_{391}$
- The public key pk is $(391, 3)$
- The secret key sk is $(391, 235)$

To encrypt $m = 158 \in \mathbb{Z}^*_{391}$:

- Compute $c = 158^3 \bmod 391 = (158^2 \bmod 391) \cdot 158 \bmod 391 \quad = 331 \cdot 158 \bmod 391 = 295$

To decrypt $c = 295$:

- Compute $m = 295^{235} \bmod 391$

RSA-Based Public Key Encryption: Example

We reduce the result modulo 295 after every product

$$295^{235} \bmod 391 \quad = (295^{117} \bmod 391)^2 \cdot 295 \bmod 391$$

RSA-Based Public Key Encryption: Example

We reduce the result modulo 295 after every product

$$295^{235} \bmod 391 \;= (295^{117} \bmod 391)^2 \cdot 295 \bmod 391$$

$$295^{117} \bmod 391 = (295^{58} \bmod 391)^2 \cdot 295 \bmod 391$$

RSA-Based Public Key Encryption: Example

We reduce the result modulo 295 after every product

$$295^{235} \bmod 391 \;\; = (295^{117} \bmod 391)^2 {\cdot} 295 \bmod 391$$

$$295^{117} \bmod 391 = (295^{58} \bmod 391)^2 \cdot 295 \bmod 391$$

$$295^{58} \bmod 391 = (295^{29} \bmod 391)^2 \bmod 391$$

RSA-Based Public Key Encryption: Example

We reduce the result modulo 295 after every product

$$295^{235} \bmod 391 = (295^{117} \bmod 391)^2 \cdot 295 \bmod 391$$

$$295^{117} \bmod 391 = (295^{58} \bmod 391)^2 \cdot 295 \bmod 391$$

$$295^{58} \bmod 391 = (295^{29} \bmod 391)^2 \bmod 391$$

$$295^{29} \bmod 391 = (295^{14} \bmod 391)^2 \cdot 295 \bmod 391$$

$$295^{14} \bmod 391 = (295^{7} \bmod 391)^2 \bmod 391$$

$$295^{7} \bmod 391 = (295^{3} \bmod 391)^2 \cdot 295 \bmod 391$$

$$295^{3} \bmod 391 = (295 \bmod 391)^2 \cdot 295 \bmod 391$$

RSA-Based Public Key Encryption: Example

We reduce the result modulo 295 after every product

$$295^{235} \bmod 391 = (295^{117} \bmod 391)^2 \cdot 295 \bmod 391$$

$$295^{117} \bmod 391 = (295^{58} \bmod 391)^2 \cdot 295 \bmod 391$$

$$295^{58} \bmod 391 = (295^{29} \bmod 391)^2 \bmod 391$$

$$295^{29} \bmod 391 = (295^{14} \bmod 391)^2 \cdot 295 \bmod 391$$

$$295^{14} \bmod 391 = (295^{7} \bmod 391)^2 \bmod 391$$

$$295^{7} \bmod 391 = (295^{3} \bmod 391)^2 \cdot 295 \bmod 391$$

$$295^{3} \bmod 391 = (295 \bmod 391)^2 \cdot 295 \bmod 391 \qquad = 97$$

RSA-Based Public Key Encryption: Example

We reduce the result modulo 295 after every product

$$
\begin{aligned}
295^{235} \bmod 391 &= (295^{117} \bmod 391)^2 \cdot 295 \bmod 391 &&= 245^2 \cdot 295 \bmod 391 &&= 158 \\
295^{117} \bmod 391 &= (295^{58} \bmod 391)^2 \cdot 295 \bmod 391 &&= 202^2 \cdot 295 \bmod 391 &&= 245 \\
295^{58} \bmod 391 &= (295^{29} \bmod 391)^2 \bmod 391 &&= 61^2 \bmod 391 &&= 202 \\
295^{29} \bmod 391 &= (295^{14} \bmod 391)^2 \cdot 295 \bmod 391 &&= 179^2 \cdot 295 \bmod 391 &&= 61 \\
295^{14} \bmod 391 &= (295^{7} \bmod 391)^2 \bmod 391 &&= 337^2 \bmod 391 &&= 179 \\
295^{7} \bmod 391 &= (295^{3} \bmod 391)^2 \cdot 295 \bmod 391 &&= 97^2 \cdot 295 \bmod 391 &&= 337 \\
295^{3} \bmod 391 &= (295 \bmod 391)^2 \cdot 295 \bmod 391 &&= 97
\end{aligned}
$$

RSA-Based Public Key Encryption: Example

We reduce the result modulo 295 after every product

$$\left.\begin{aligned}
295^{235} \bmod 391 &= (295^{117} \bmod 391)^2 \cdot 295 \bmod 391 && = 245^2 \cdot 295 \bmod 391 && = 158\\
295^{117} \bmod 391 &= (295^{58} \bmod 391)^2 \cdot 295 \bmod 391 && = 202^2 \cdot 295 \bmod 391 && = 245\\
295^{58} \bmod 391 &= (295^{29} \bmod 391)^2 \bmod 391 && = 61^2 \bmod 391 && = 202\\
295^{29} \bmod 391 &= (295^{14} \bmod 391)^2 \cdot 295 \bmod 391 && = 179^2 \cdot 295 \bmod 391 && = 61\\
295^{14} \bmod 391 &= (295^{7} \bmod 391)^2 \bmod 391 && = 337^2 \bmod 391 && = 179\\
295^{7} \bmod 391 &= (295^{3} \bmod 391)^2 \cdot 295 \bmod 391 && = 97^2 \cdot 295 \bmod 391 && = 337\\
295^{3} \bmod 391 &= (295 \bmod 391)^2 \cdot 295 \bmod 391 && = 97 &&
\end{aligned}\right\} \begin{array}{c} \approx \log_2 d \\ \leq \log_2 N \\ \text{levels of} \\ \text{recursion} \end{array}$$

RSA-Based Public Key Encryption: Example

We reduce the result modulo 295 after every product

$$\left.\begin{aligned}
295^{235} \bmod 391 &= (295^{117} \bmod 391)^2 \cdot 295 \bmod 391 && = 245^2 \cdot 295 \bmod 391 && = 158\\
295^{117} \bmod 391 &= (295^{58} \bmod 391)^2 \cdot 295 \bmod 391 && = 202^2 \cdot 295 \bmod 391 && = 245\\
295^{58} \bmod 391 &= (295^{29} \bmod 391)^2 \bmod 391 && = 61^2 \bmod 391 && = 202\\
295^{29} \bmod 391 &= (295^{14} \bmod 391)^2 \cdot 295 \bmod 391 && = 179^2 \cdot 295 \bmod 391 && = 61\\
295^{14} \bmod 391 &= (295^{7} \bmod 391)^2 \bmod 391 && = 337^2 \bmod 391 && = 179\\
295^{7} \bmod 391 &= (295^{3} \bmod 391)^2 \cdot 295 \bmod 391 && = 97^2 \cdot 295 \bmod 391 && = 337\\
295^{3} \bmod 391 &= (295 \bmod 391)^2 \cdot 295 \bmod 391 && = 97
\end{aligned}\right\}\begin{array}{c}\approx \log_2 d\\ \leq \log_2 N\\ \text{levels of}\\ \text{recursion}\end{array}$$

$m = 295^{235} \bmod 391 = 158$

Security of Plain RSA

Is plain RSA secure (under the RSA assumption)?

Observation 1:

- The RSA assumption is defined with respect to the RSA-inv experiment

Security of Plain RSA

Is plain RSA secure (under the RSA assumption)?

Observation 1:

- The RSA assumption is defined with respect to the RSA-inv experiment
- In the RSA-inv experiment the adversary wins by computing the e-th root of a **random group element** $x \in Z_N^*$

Security of Plain RSA

Is plain RSA secure (under the RSA assumption)?

Observation 1:

- The RSA assumption is defined with respect to the RSA-inv experiment
- In the RSA-inv experiment the adversary wins by computing the e-th root of a **random group element** $x \in Z_N^*$
- If m is a message chosen u.a.r. then, since $f_e(x) = x^e$ is a permutation, m^e is also uniformly distributed in Z_N^*

Security of Plain RSA

Is plain RSA secure (under the RSA assumption)?

Observation 1:

- The RSA assumption is defined with respect to the RSA-inv experiment
- In the RSA-inv experiment the adversary wins by computing the e-th root of a **random group element** $x \in Z_N^*$
- If m is a message chosen u.a.r. then, since $f_e(x) = x^e$ is a permutation, m^e is also uniformly distributed in Z_N^*

- In general m is not a random message!

Security of Plain RSA

Is plain RSA secure (under the RSA assumption)?

Observation 1:

- The RSA assumption is defined with respect to the RSA-inv experiment
- In the RSA-inv experiment the adversary wins by computing the e-th root of a **random group element** $x \in Z_N^*$
- If m is a message chosen u.a.r. then, since $f_e(x) = x^e$ is a permutation, m^e is also uniformly distributed in Z_N^*
- In general m is not a random message! $\implies$ We can't rely on the RSA assumption.

Security of Plain RSA

Is plain RSA secure (under the RSA assumption)?

Observation 1:

- The RSA assumption is defined with respect to the RSA-inv experiment
- In the RSA-inv experiment the adversary wins by computing the e-th root of a **random group element** $x \in Z_N^*$
- If m is a message chosen u.a.r. then, since $f_e(x) = x^e$ is a permutation, m^e is also uniformly distributed in Z_N^* ☺
- In general m is not a random message! $\implies$ We can't rely on the RSA assumption. ☹

Observation 2:

- The plain RSA scheme is **deterministic**!

Security of Plain RSA

Is plain RSA secure (under the RSA assumption)?

Observation 1:

- The RSA assumption is defined with respect to the RSA-inv experiment
- In the RSA-inv experiment the adversary wins by computing the e-th root of a **random group element** $x \in Z_N^*$
- If m is a message chosen u.a.r. then, since $f_e(x) = x^e$ is a permutation, m^e is also uniformly distributed in Z_N^*
- In general m is not a random message! $\implies$ We can't rely on the RSA assumption.

Observation 2:

- The plain RSA scheme is **deterministic**!
- It can't be CPA-secure!

Security of Plain RSA

Is plain RSA secure (under the RSA assumption)?

Observation 1:

- The RSA assumption is defined with respect to the RSA-inv experiment
- In the RSA-inv experiment the adversary wins by computing the e-th root of a **random group element** $x \in Z_N^*$
- If m is a message chosen u.a.r. then, since $f_e(x) = x^e$ is a permutation, m^e is also uniformly distributed in Z_N^* ☺
- In general m is not a random message! $\implies$ We can't rely on the RSA assumption. ☹

Observation 2:

- The plain RSA scheme is **deterministic**!
- It can't be CPA-secure!

(and since CPA-security and EAV-security concide in the public-key world, it is not even secure against an eavesdropper)

Security of Plain RSA

Is plain RSA secure (under the RSA assumption)?

Observation 1:

- The RSA assumption is defined with respect to the RSA-inv experiment
- In the RSA-inv experiment the adversary wins by computing the e-th root of a **random group element** $x \in Z_N^*$
- If m is a message chosen u.a.r. then, since $f_e(x) = x^e$ is a permutation, m^e is also uniformly distributed in Z_N^*
- In general m is not a random message! $\implies$ We can't rely on the RSA assumption.

Observation 2:

- The plain RSA scheme is **deterministic**!
- It can't be CPA-secure!

(and since CPA-security and EAV-security concide in the public-key world, it is not even secure against an eavesdropper)

Plain RSA should never be used!

Attacks on Plain RSA: Better than bruteforce

Suppose that plain-RSA is used to encrypt a random group element $m \in \mathbb{Z}_N^*$ with $N \approx 2^\eta$

- Clearly we can recover m by trying all the $\approx 2^\eta$ possible choices with a bruteforce attack

Attacks on Plain RSA: Better than bruteforce

Suppose that plain-RSA is used to encrypt a random group element $m \in \mathbb{Z}_N^*$ with $N \approx 2^\eta$

- Clearly we can recover m by trying all the $\approx 2^\eta$ possible choices with a bruteforce attack
- One could hope that the **best-known attack** is not significantly better than the bruteforce attack

Attacks on Plain RSA: Better than bruteforce

Suppose that plain-RSA is used to encrypt a random group element $m \in \mathbb{Z}_N^*$ with $N \approx 2^\eta$

- Clearly we can recover m by trying all the $\approx 2^\eta$ possible choices with a bruteforce attack
- One could hope that the **best-known attack** is not significantly better than the bruteforce attack
- There is an attack that takes time $\approx \sqrt{N}$, i.e., $\approx 2^{\frac{\eta}{2}}$

Attacks on Plain RSA: Better than bruteforce

Suppose that plain-RSA is used to encrypt a random group element $m \in \mathbb{Z}_N^*$ with $N \approx 2^\eta$

- Clearly we can recover m by trying all the $\approx 2^\eta$ possible choices with a bruteforce attack
- One could hope that the **best-known attack** is not significantly better than the bruteforce attack
- There is an attack that takes time $\approx \sqrt{N}$, i.e., $\approx 2^{\frac{\eta}{2}}$

Let $P_\alpha(\eta)$ denote the probability that a η-bit message m can be written as $m = m_1 \cdot m_2$ with $m_1, m_2 \leq 2^{\alpha\eta}$.

$$\lim_{\eta\to\infty} \inf P_\alpha(\eta) \geq \ln(2\alpha)$$

Attacks on Plain RSA: Better than bruteforce

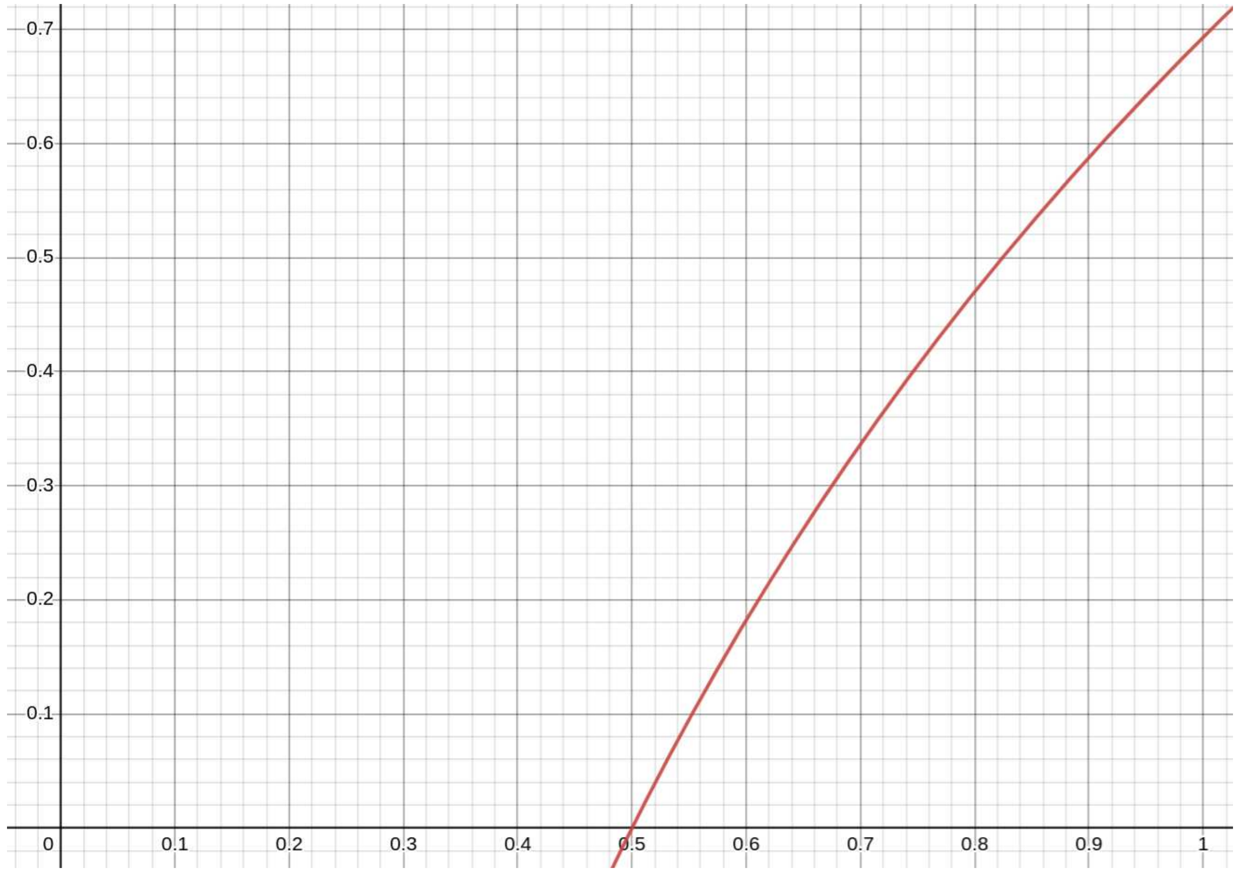
Suppose that plain-RSA is used to encrypt a random group element $m \in \mathbb{Z}_N^*$ with $N \approx 2^\eta$

- Clearly we can recover m by trying all the $\approx 2^\eta$ possible choices with a bruteforce attack
- One could hope that the **best-known attack** is not significantly better than the bruteforce attack
- There is an attack that takes time $\approx \sqrt{N}$, i.e., $\approx 2^{\frac{\eta}{2}}$

Let $P_\alpha(\eta)$ denote the probability that a η-bit message m can be written as $m = m_1 \cdot m_2$ with $m_1, m_2 \leq 2^{\alpha\eta}$.

$$\lim_{\eta\to\infty} \inf P_\alpha(\eta) \geq \ln(2\alpha)$$

For any constant $\alpha > \frac{1}{2}$, $P_\alpha(\eta)$ is a positive constant!



Dan Boneh, Antoine Joux & Phong Q. Nguyen.
Why Textbook ElGamal and RSA Encryption Are Insecure

Attacks on Plain RSA: Better than bruteforce

Suppose that plain-RSA is used to encrypt a random group element $m \in \mathbb{Z}_N^*$ with $N \approx 2^\eta$

- Clearly we can recover m by trying all the $\approx 2^\eta$ possible choices with a bruteforce attack
- One could hope that the **best-known attack** is not significantly better than the bruteforce attack
- There is an attack that takes time $\approx \sqrt{N}$, i.e., $\approx 2^{\frac{\eta}{2}}$

Let $P_\alpha(\eta)$ denote the probability that a η-bit message m can be written as $m = m_1 \cdot m_2$ with $m_1, m_2 \leq 2^{\alpha\eta}$.

$$\lim_{\eta\to\infty} \inf P_\alpha(\eta) \geq \ln(2\alpha)$$

For any constant $\alpha > \frac{1}{2}$, $P_\alpha(\eta)$ is a positive constant!

Table 1. Experimental probabilities of splitting into two factors.

Bit-length m	m_1	m_2	Probability
40	20	20	18%
	21	21	32%
	22	22	39%
	20	25	50%
64	32	32	18%
	33	33	29%
	34	34	35%
	30	36	40%

Dan Boneh, Antoine Joux & Phong Q. Nguyen.
Why Textbook ElGamal and RSA Encryption Are Insecure

Attacks on Plain RSA: Better than bruteforce

$$c = m^e = (m_1 \cdot m_2)^e = m_1^e \cdot m_2^e$$

Attacks on Plain RSA: Better than bruteforce

$c = m^e = (m_1 \cdot m_2)^e = m_1^e \cdot m_2^e \qquad \implies \qquad m_1^e = c \cdot (m_2^e)^{-1}$

Attacks on Plain RSA: Better than bruteforce

$$c = m^e = (m_1 \cdot m_2)^e = m_1^e \cdot m_2^e \quad \Longrightarrow \quad m_1^e = c \cdot (m_2^e)^{-1} \longrightarrow$$

Call this (unknown) element x

Attacks on Plain RSA: Better than bruteforce

$c = m^e = (m_1 \cdot m_2)^e = m_1^e \cdot m_2^e \quad \Longrightarrow \quad m_1^e = c \cdot (m_2^e)^{-1}$

Call this (unknown) element x

Input: c

Time

- $L \leftarrow$ empty list
- For $\widetilde{m}_2 = 1, 2, \ldots, 2^{\alpha\eta}$: (guess m_2) $\approx 2^{\alpha n}$
  - Compute $\widetilde{x} = c \cdot (\widetilde{m}_2^e)^{-1}$ (compute what the value of x would be if the guess was correct)
  - Append $(\widetilde{m}_2, \widetilde{x})$ to L

Attacks on Plain RSA: Better than bruteforce

$c = m^e = (m_1 \cdot m_2)^e = m_1^e \cdot m_2^e \quad \Longrightarrow \quad m_1^e = c \cdot (m_2^e)^{-1}$

Call this (unknown) element x

Input: c

Time

- $L \leftarrow$ empty list
- For $\widetilde{m}_2 = 1, 2, \ldots, 2^{\alpha\eta}$: (guess m_2) $\approx 2^{\alpha n}$
  - Compute $\widetilde{x} = c \cdot (\widetilde{m}_2^e)^{-1}$ (compute what the value of x would be if the guess was correct)
  - Append $(\widetilde{m}_2, \widetilde{x})$ to L
- Sort the pairs in L w.r.t. their second component

Attacks on Plain RSA: Better than bruteforce

$$c = m^e = (m_1 \cdot m_2)^e = m_1^e \cdot m_2^e \quad \Longrightarrow \quad m_1^e = c \cdot (m_2^e)^{-1} \longrightarrow$$ Call this (unknown) element x

Input: c

Time

- $L \leftarrow$ empty list
- For $\widetilde{m}_2 = 1, 2, \ldots, 2^{\alpha\eta}$: (guess m_2) $\approx 2^{\alpha n}$
  - Compute $\widetilde{x} = c \cdot (\widetilde{m}_2^e)^{-1}$ (compute what the value of x would be if the guess was correct)
  - Append $(\widetilde{m}_2, \widetilde{x})$ to L
- Sort the pairs in L w.r.t. their second component $\approx \alpha n \cdot 2^{\alpha n}$
- For $\widetilde{m}_1 = 1, 2, \ldots, 2^{\alpha\eta}$: (guess m_1) $\approx \alpha n \cdot 2^{\alpha n}$
  - Compute $\widetilde{x} = \widetilde{m}_1^e$ (compute what the value of x would be if the guess was correct)
  - If there is $\widetilde{m}_2$ s.t. $(\widetilde{m}_2, \widetilde{x}) \in L$: (use binary search) $\approx \alpha n$
    - Return $m = \widetilde{m}_1 \cdot \widetilde{m}_2$
- Return "failure"

Attacks on Plain RSA: Better than bruteforce

$$c = m^e = (m_1 \cdot m_2)^e = m_1^e \cdot m_2^e \implies m_1^e = c \cdot (m_2^e)^{-1}$$

Call this (unknown) element x

Input: c

Time

- $L \leftarrow$ empty list
- For $\widetilde{m}_2 = 1, 2, \ldots, 2^{\alpha\eta}$: (guess m_2) $\approx 2^{\alpha n}$
  - Compute $\widetilde{x} = c \cdot (\widetilde{m}_2^e)^{-1}$ (compute what the value of x would be if the guess was correct)
  - Append $(\widetilde{m}_2, \widetilde{x})$ to L
- Sort the pairs in L w.r.t. their second component $\approx \alpha n \cdot 2^{\alpha n}$
- For $\widetilde{m}_1 = 1, 2, \ldots, 2^{\alpha\eta}$: (guess m_1) $\approx \alpha n \cdot 2^{\alpha n}$
  - Compute $\widetilde{x} = \widetilde{m}_1^e$ (compute what the value of x would be if the guess was correct)
  - If there is $\widetilde{m}_2$ s.t. $(\widetilde{m}_2, \widetilde{x}) \in L$: (use binary search) $\approx \alpha n$
    - Return $m = \widetilde{m}_1 \cdot \widetilde{m}_2$
- Return "failure"

Success probability: $P_\alpha(\eta)$

Time: $\approx \alpha n \cdot 2^{\alpha n}$

Attacks on Plain RSA: short messages and small exponent

Suppose that $m \leq \sqrt[e]{N}$ (here operations are performed over the reals)

- This happens when e is small (recall that $e = 3$ is a common choice)...
- ...and m is short

Attacks on Plain RSA: short messages and small exponent

Suppose that $m \leq \sqrt[e]{N}$ (here operations are performed over the reals)

- This happens when e is small (recall that $e = 3$ is a common choice)...
- ...and m is short
- For example when N has 2048 bits, $e = 3$, and m has ≤ 85 bytes

Attacks on Plain RSA: short messages and small exponent

Suppose that $m \leq \sqrt[e]{N}$ (here operations are performed over the reals)

- This happens when e is small (recall that $e = 3$ is a common choice)...
- ...and m is short
- For example when N has 2048 bits, $e = 3$, and m has ≤ 85 bytes

Then $m^e \leq N$ and hence $c = m^e \bmod N = m^e$

No modular reduction!

Attacks on Plain RSA: short messages and small exponent

Suppose that $m \leq \sqrt[e]{N}$ (here operations are performed over the reals)

- This happens when e is small (recall that $e = 3$ is a common choice)...
- ...and m is short
- For example when N has 2048 bits, $e = 3$, and m has ≤ 85 bytes

Then $m^e \leq N$ and hence $c = m^e \bmod N = m^e$

No modular reduction!

- m can be recovered from c by computing $\sqrt[e]{c}$ **over the reals**!

Attacks on Plain RSA: short messages and small exponent

Suppose that $m \leq \sqrt[e]{N}$ (here operations are performed over the reals)

- This happens when e is small (recall that $e = 3$ is a common choice)...
- ...and m is short
- For example when N has 2048 bits, $e = 3$, and m has ≤ 85 bytes

Then $m^e \leq N$ and hence $c = m^e \bmod N = m^e$

No modular reduction!

- m can be recovered from c by computing $\sqrt[e]{c}$ **over the reals**!
- This requires polynomial-time!

Attacks on Plain RSA: partially-known messages

Theorem: Let $p(x)$ be a polynomial of degree e. All x such that $p(x) = 0 \bmod N$ and $|x| \leq N^{1/e}$ can be found in time poly$(\log N, e)$.

Attacks on Plain RSA: partially-known messages

Theorem: Let $p(x)$ be a polynomial of degree e. All x such that $p(x) = 0 \bmod N$ and $|x| \leq N^{1/e}$ can be found in time poly$(\log N, e)$.

Suppose that the sender encrypts a message $m = m_1 \| m_2$, where m_1 is known to the attacker and m_2 has k bits

$$m = m_1 \cdot 2^k + m_2$$

Attacks on Plain RSA: partially-known messages

Theorem: Let $p(x)$ be a polynomial of degree e. All x such that $p(x) = 0 \bmod N$ and $|x| \leq N^{1/e}$ can be found in time poly$(\log N, e)$.

Suppose that the sender encrypts a message $m = m_1 \| m_2$, where m_1 is known to the attacker and m_2 has k bits

$$m = m_1 \cdot 2^k + m_2$$

After encryption:

$$c = m^e = (m_1 \cdot 2^k + m_2)^e$$

Attacks on Plain RSA: partially-known messages

Theorem: Let $p(x)$ be a polynomial of degree e. All x such that $p(x) = 0 \bmod N$ and $|x| \leq N^{1/e}$ can be found in time poly$(\log N, e)$.

Suppose that the sender encrypts a message $m = m_1 \| m_2$, where m_1 is known to the attacker and m_2 has k bits

$$m = m_1 \cdot 2^k + m_2$$

After encryption:

$$c = m^e = (m_1 \cdot 2^k + m_2)^e$$

Therefore the message m_2 satisfies:

$$(m_1 \cdot 2^k + m_2)^e - c = 0$$

Attacks on Plain RSA: partially-known messages

Theorem: Let $p(x)$ be a polynomial of degree e. All x such that $p(x) = 0 \bmod N$ and $|x| \leq N^{1/e}$ can be found in time poly$(\log N, e)$.

Suppose that the sender encrypts a message $m = m_1 \| m_2$, where m_1 is known to the attacker and m_2 has k bits

$$m = m_1 \cdot 2^k + m_2$$

After encryption:

$$c = m^e = (m_1 \cdot 2^k + m_2)^e$$

This is a polynomial of degree e (w.r.t. m_2), and we are only interested in solutions such that $m_2 < 2^k$

Therefore the message m_2 satisfies:

$$(m_1 \cdot 2^k + m_2)^e - c = 0$$

Attacks on Plain RSA: partially-known messages

Theorem: Let $p(x)$ be a polynomial of degree e. All x such that $p(x) = 0 \bmod N$ and $|x| \leq N^{1/e}$ can be found in time poly$(\log N, e)$.

Suppose that the sender encrypts a message $m = m_1 \| m_2$, where m_1 is known to the attacker and m_2 has k bits

$$m = m_1 \cdot 2^k + m_2$$

After encryption:

$$c = m^e = (m_1 \cdot 2^k + m_2)^e$$

Therefore the message m_2 satisfies:

$$(m_1 \cdot 2^k + m_2)^e - c = 0$$

This is a polynomial of degree e (w.r.t. m_2), and we are only interested in solutions such that $m_2 < 2^k$

If $2^k \leq N^{1/e}$ and e is small, the above theorem allows us to list all candidates for m_2 in polynomial-time

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Suppose that e recipients all use the same exponent and have public keys:

$$pk_1 = (N_1, e) \qquad pk_2 = (N_2, e) \qquad \cdots \qquad pk_e = (N_e, e)$$

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Suppose that e recipients all use the same exponent and have public keys:

$pk_1 = (N_1, e)$ $\quad pk_2 = (N_2, e)$ $\quad \cdots$ $\quad pk_e = (N_e, e)$

An eavesdropper sees:

$c_1 = m^e \bmod N_1$ $\quad c_2 = m^e \bmod N_2$ $\quad \cdots$ $\quad c_e = m^e \bmod N_e$

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Suppose that e recipients all use the same exponent and have public keys:

$$pk_1 = (N_1, e) \qquad pk_2 = (N_2, e) \qquad \cdots \qquad pk_e = (N_e, e)$$

An eavesdropper sees:

$$c_1 = m^e \bmod N_1 \qquad c_2 = m^e \bmod N_2 \qquad \cdots \qquad c_e = m^e \bmod N_e$$

If $gcd(N_i, N_j) \neq 1$ for some i, j with $i \neq j$, then we can factor N_i and N_j. Therefore we assume that all N_is are pairwise coprime.

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Suppose that e recipients all use the same exponent and have public keys:

$$pk_1 = (N_1, e) \qquad pk_2 = (N_2, e) \qquad \cdots \qquad pk_e = (N_e, e)$$

An eavesdropper sees:

$$c_1 = m^e \bmod N_1 \qquad c_2 = m^e \bmod N_2 \qquad \cdots \qquad c_e = m^e \bmod N_e$$

If $gcd(N_i, N_j) \neq 1$ for some i, j with $i \neq j$, then we can factor N_i and N_j. Therefore we assume that all N_is are pairwise coprime.

Consider the system $\begin{cases} c \equiv c_1 \pmod{N_1} \\ \vdots \\ c \equiv c_e \pmod{N_e} \end{cases}$

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Suppose that e recipients all use the same exponent and have public keys:

$pk_1 = (N_1, e)$ $\quad pk_2 = (N_2, e)$ $\quad \cdots$ $\quad pk_e = (N_e, e)$

An eavesdropper sees:

$c_1 = m^e \bmod N_1$ $\quad c_2 = m^e \bmod N_2$ $\quad \cdots$ $\quad c_e = m^e \bmod N_e$

If $gcd(N_i, N_j) \neq 1$ for some i, j with $i \neq j$, then we can factor N_i and N_j. Therefore we assume that all N_is are pairwise coprime.

Consider the system $\begin{cases} c \equiv c_1 \pmod{N_1} \\ \vdots \\ c \equiv c_e \pmod{N_e} \end{cases}$

Notice that $c = m^e$ satisfies all equations.

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Suppose that e recipients all use the same exponent and have public keys:

$pk_1 = (N_1, e)$ $\quad$ $pk_2 = (N_2, e)$ $\quad \cdots \quad$ $pk_e = (N_e, e)$

An eavesdropper sees:

$c_1 = m^e \bmod N_1$ $\quad$ $c_2 = m^e \bmod N_2$ $\quad \cdots \quad$ $c_e = m^e \bmod N_e$

If $gcd(N_i, N_j) \neq 1$ for some i, j with $i \neq j$, then we can factor N_i and N_j. Therefore we assume that all N_is are pairwise coprime.

Consider the system $\begin{cases} c \equiv c_1 \pmod{N_1} \\ \vdots \\ c \equiv c_e \pmod{N_e} \end{cases}$

Notice that $c = m^e$ satisfies all equations.

Moreover, $0 \leq m^e < \prod_{i=1}^{e} N_i$

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Suppose that e recipients all use the same exponent and have public keys:

$pk_1 = (N_1, e)$ $\qquad$ $pk_2 = (N_2, e)$ $\qquad \cdots \qquad$ $pk_e = (N_e, e)$

An eavesdropper sees:

$c_1 = m^e \bmod N_1$ $\qquad$ $c_2 = m^e \bmod N_2$ $\qquad \cdots \qquad$ $c_e = m^e \bmod N_e$

If $gcd(N_i, N_j) \neq 1$ for some i, j with $i \neq j$, then we can factor N_i and N_j. Therefore we assume that all N_is are pairwise coprime.

Chinese Remainder Theorem: If the n_i are pairwise coprime then the system

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ \vdots \\ x \equiv a_k \pmod{n_k} \end{cases}$$

has a unique solution x^* s.t. $0 \leq x^* < \prod_{i=1}^{k} n_i$. Moreover, all solutions are congruent modulo $\prod_{i=1}^{k} n_i$.

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Suppose that e recipients all use the same exponent and have public keys:

$pk_1 = (N_1, e)$ $\quad$ $pk_2 = (N_2, e)$ $\quad$ $\cdots$ $\quad$ $pk_e = (N_e, e)$

An eavesdropper sees:

$c_1 = m^e \bmod N_1$ $\quad$ $c_2 = m^e \bmod N_2$ $\quad$ $\cdots$ $\quad$ $c_e = m^e \bmod N_e$

If $gcd(N_i, N_j) \neq 1$ for some i, j with $i \neq j$, then we can factor N_i and N_j. Therefore we assume that all N_is are pairwise coprime.

Consider the system
$$\begin{cases} c \equiv c_1 \pmod{N_1} \\ \vdots \\ c \equiv c_e \pmod{N_e} \end{cases}$$

Notice that $c = m^e$ satisfies all equations.

Moreover, $0 \leq m^e < \prod_{i=1}^{e} N_i$

The solution whose existence is guaranteed by the Chinese remainder theorem is exactly $c^* = m^e$

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Suppose that e recipients all use the same exponent and have public keys:

$pk_1 = (N_1, e) \qquad pk_2 = (N_2, e) \qquad \cdots \qquad pk_e = (N_e, e)$

An eavesdropper sees:

$c_1 = m^e \bmod N_1 \qquad c_2 = m^e \bmod N_2 \qquad \cdots \qquad c_e = m^e \bmod N_e$

If $gcd(N_i, N_j) \neq 1$ for some i, j with $i \neq j$, then we can factor N_i and N_j. Therefore we assume that all N_is are pairwise coprime.

Consider the system $\begin{cases} c \equiv c_1 \pmod{N_1} \\ \vdots \\ c \equiv c_e \pmod{N_e} \end{cases}$

Notice that $c = m^e$ satisfies all equations.

Moreover, $0 \leq m^e < \prod_{i=1}^{e} N_i$

No modular reduction!

The solution whose existence is guaranteed by the Chinese remainder theorem is exactly $c^* = m^e$

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Suppose that e recipients all use the same exponent and have public keys:

$pk_1 = (N_1, e) \qquad pk_2 = (N_2, e) \qquad \cdots \qquad pk_e = (N_e, e)$

An eavesdropper sees:

$c_1 = m^e \bmod N_1 \qquad c_2 = m^e \bmod N_2 \qquad \cdots \qquad c_e = m^e \bmod N_e$

If $gcd(N_i, N_j) \neq 1$ for some i, j with $i \neq j$, then we can factor N_i and N_j. Therefore we assume that all N_is are pairwise coprime.

Consider the system $\begin{cases} c \equiv c_1 \pmod{N_1} \\ \vdots \\ c \equiv c_e \pmod{N_e} \end{cases}$

Notice that $c = m^e$ satisfies all equations.

Moreover, $0 \leq m^e < \prod_{i=1}^{e} N_i$

No modular reduction!

The solution whose existence is guaranteed by the Chinese remainder theorem is exactly $c^* = m^e$

This is solution can be found in polynomial-time!

Attacks on Plain RSA: encrypting a message multiple times

Consider a sender that encrypts the same message m for multiple recipients

Suppose that e recipients all use the same exponent and have public keys:

$pk_1 = (N_1, e) \qquad pk_2 = (N_2, e) \qquad \cdots \qquad pk_e = (N_e, e)$

An eavesdropper sees:

$c_1 = m^e \bmod N_1 \qquad c_2 = m^e \bmod N_2 \qquad \cdots \qquad c_e = m^e \bmod N_e$

If $gcd(N_i, N_j) \neq 1$ for some i, j with $i \neq j$, then we can factor N_i and N_j. Therefore we assume that all N_is are pairwise coprime.

Consider the system $\begin{cases} c \equiv c_1 \pmod{N_1} \\ \vdots \\ c \equiv c_e \pmod{N_e} \end{cases}$

Notice that $c = m^e$ satisfies all equations.

Moreover, $0 \leq m^e < \prod_{i=1}^{e} N_i$

No modular reduction!

The solution whose existence is guaranteed by the Chinese remainder theorem is exactly $c^* = m^e$

This is solution can be found in polynomial-time!

To recover m we just need to compute $\sqrt[e]{c^*}$ (over the reals!)