ARIEL UNIVERSITY

אוניברסיטת אריאל בשומרון

Foundations of Modern Cryptography

Edward A. Hirsch

https://edwardahirsch.github.io/edwardahirsch

Neapolis University Pafos

Lecture 11: January 9, 2025

Lecture Plan

- A complete public-key cryptosystem.
- Limitations of modern cryptography.

If the screen seems frozen and I do not respond,
please call me in Telegram.

Why do we believe our constructions are secure?

- We proved that no adversary can break our schemes provided the primitives are secure.
  - Proved via **reductions**: if we break a scheme for security parameter n with probability p, then we break a primitive for related n' with related probability p'.

Why do we believe our constructions are secure?

- We proved that no adversary can break our schemes provided the primitives are secure.
  - Proved via **reductions**: if we break a scheme for security parameter n with probability p, then we break a primitive for related n' with related probability p'.
- What about the primitives?
  - One-way functions: Levin's universal one-way function!
  - Trapdoor permutation families:

Why do we believe our constructions are secure?

- We proved that no adversary can break our schemes provided the primitives are secure.
  - Proved via **reductions**: if we break a scheme for security parameter n with probability p, then we break a primitive for related n' with related probability p'.
- What about the primitives?
  - One-way functions: Levin's universal one-way function!
  - Trapdoor permutation families:
    - Hmm, at least we can construct a complete (universal) public-key cryptosystem! (Coming.)

Why do we believe our constructions are secure?

- We proved that no adversary can break our schemes provided the primitives are secure.
  - Proved via **reductions**: if we break a scheme for security parameter n with probability p, then we break a primitive for related n' with related probability p'.
- What about the primitives?
  - One-way functions: Levin's universal one-way function!
  - Trapdoor permutation families:
    - Hmm, at least we can construct a complete (universal) public-key cryptosystem! (Coming.)
  - Oblivious transfer: does not seem likely.
    - Yet universal schemes for secure function evaluation are possible
      [Jain, Manohar, Sahai, EUROCRYPT-2020] — disclaimer: I did not read this paper.

Why do we believe our constructions are secure?

- We proved that no adversary can break our schemes provided the primitives are secure.
  - Proved via **reductions**: if we break a scheme for security parameter n with probability p, then we break a primitive for related n' with related probability p'.
- What about the primitives?
  - One-way functions: Levin's universal one-way function!
  - Trapdoor permutation families:
    - Hmm, at least we can construct a complete (universal) public-key cryptosystem! (Coming.)
  - Oblivious transfer: does not seem likely.
    - Yet universal schemes for secure function evaluation are possible
      [Jain, Manohar, Sahai, EUROCRYPT-2020] — disclaimer: I did not read this paper.
  - Enhanced trapdoor permutations (with errors)? Check it yourself.

A complete (universal) public-key cryptosystem for 1 bit

The hardest to break cryptosystem!

A complete (universal) public-key cryptosystem for 1 bit

The hardest to break cryptosystem!

- Allow decryption errors:
  - G generates a pair of keys (public key $\mathtt{pk}$, secret/private key $\mathtt{sk}$): $G\colon (1^n, r_g) \mapsto (\mathtt{pk}, \mathtt{sk})$.
  - An adversary is unable to decrypt (without $\mathtt{sk}$) a random $\mathtt{msg}$ with non-negligible probability:

$$\mathbf{Pr}\{A(\mathtt{pk}, E(\mathtt{pk}, \mathtt{msg})) = \mathtt{msg}\} < \varepsilon(n).$$

  - Legitimate parties decrypt correctly (most of the time): $\forall\, \mathtt{msg}$

$$\mathbf{Pr}\{D(\mathtt{sk}, E(\mathtt{pk}, \mathtt{msg})) = \mathtt{msg}\} \geqslant 0.9. \qquad \text{(CORR)}$$

A complete (universal) public-key cryptosystem for 1 bit

The hardest to break cryptosystem!

- Allow decryption errors:
  - G generates a pair of keys (public key $\mathtt{pk}$, secret/private key $\mathtt{sk}$): $G\colon (1^n, r_g) \mapsto (\mathtt{pk}, \mathtt{sk})$.
  - An adversary is unable to decrypt (without $\mathtt{sk}$) a random $\mathtt{msg}$ with non-negligible probability:

$$\mathbf{Pr}\{A(\mathtt{pk}, E(\mathtt{pk}, \mathtt{msg})) = \mathtt{msg}\} < \varepsilon(n).$$

  - Legitimate parties decrypt correctly (most of the time): $\forall\, \mathtt{msg}$

$$\mathbf{Pr}\{D(\mathtt{sk}, E(\mathtt{pk}, \mathtt{msg})) = \mathtt{msg}\} \geqslant 0.9. \qquad \text{(CORR)}$$

- Try all possible cryptosystems (G_i, E_i, D_i)!

A complete (universal) public-key cryptosystem for 1 bit

The hardest to break cryptosystem!

- Allow decryption errors:
  - G generates a pair of keys (public key $\mathtt{pk}$, secret/private key $\mathtt{sk}$): $G\colon (1^n, r_g) \mapsto (\mathtt{pk}, \mathtt{sk})$.
  - An adversary is unable to decrypt (without $\mathtt{sk}$) a random $\mathtt{msg}$ with non-negligible probability:

$$\mathbf{Pr}\{A(\mathtt{pk}, E(\mathtt{pk}, \mathtt{msg})) = \mathtt{msg}\} < \varepsilon(n).$$

  - Legitimate parties decrypt correctly (most of the time): $\forall\, \mathtt{msg}$

$$\mathbf{Pr}\{D(\mathtt{sk}, E(\mathtt{pk}, \mathtt{msg})) = \mathtt{msg}\} \geqslant 0.9. \qquad \text{(CORR)}$$

- Try all possible cryptosystems (G_i, E_i, D_i)!
- To combine k systems into one, split the bit $b = b_1 \oplus \ldots \oplus b_k$ and encrypt its "parts" b_i.

A complete (universal) public-key cryptosystem for 1 bit

The hardest to break cryptosystem!

- Allow decryption errors:
  - G generates a pair of keys (public key $\mathtt{pk}$, secret/private key $\mathtt{sk}$): $G\colon (1^n, r_g) \mapsto (\mathtt{pk}, \mathtt{sk})$.
  - An adversary is unable to decrypt (without $\mathtt{sk}$) a random $\mathtt{msg}$ with non-negligible probability:

$$\mathbf{Pr}\{A(\mathtt{pk}, E(\mathtt{pk}, \mathtt{msg})) = \mathtt{msg}\} < \varepsilon(n).$$

  - Legitimate parties decrypt correctly (most of the time): $\forall\, \mathtt{msg}$

$$\mathbf{Pr}\{D(\mathtt{sk}, E(\mathtt{pk}, \mathtt{msg})) = \mathtt{msg}\} \geqslant 0.9. \qquad \text{(CORR)}$$

- Try all possible cryptosystems (G_i, E_i, D_i)!
- To combine k systems into one, split the bit $b = b_1 \oplus \ldots \oplus b_k$ and encrypt its "parts" b_i.
- To avoid using incorrect cryptosystems, test them before using: verify (CORR).

A complete (universal) public-key cryptosystem for 1 bit

The hardest to break cryptosystem!

- Allow decryption errors:
  - G generates a pair of keys (public key $\mathtt{pk}$, secret/private key $\mathtt{sk}$): $G\colon (1^n, r_g) \mapsto (\mathtt{pk}, \mathtt{sk})$.
  - An adversary is unable to decrypt (without $\mathtt{sk}$) a random $\mathtt{msg}$ with non-negligible probability:

$$\mathbf{Pr}\{A(\mathtt{pk}, E(\mathtt{pk}, \mathtt{msg})) = \mathtt{msg}\} < \varepsilon(n).$$

  - Legitimate parties decrypt correctly (most of the time): $\forall\, \mathtt{msg}$

$$\mathbf{Pr}\{D(\mathtt{sk}, E(\mathtt{pk}, \mathtt{msg})) = \mathtt{msg}\} \geqslant \boldsymbol{\delta}. \qquad (\delta\text{-CORR})$$

- Try all possible cryptosystems (G_i, E_i, D_i)!
- To combine k systems into one, split the bit $b = b_1 \oplus \ldots \oplus b_k$ and encrypt its "parts" b_i.
- To avoid using incorrect cryptosystems, test them before using: verify (CORR).
- To avoid error accumulation, encrypt $E_i(\mathtt{pk}_i, b_i)$ many times with different $\mathtt{pk}_i$ and randomness.

From 0.9-correct to almost correct

Correctness

- Use G to generate t independent key pairs $\mathtt{pk} = (\mathtt{pk}_1, \ldots, \mathtt{pk}_t)$, $\mathtt{sk} = (\mathtt{sk}_1, \ldots, \mathtt{sk}_t)$.
- Encrypt using independent randomness:

$$E(\mathtt{pk}, b) = (E(\mathtt{pk}_1, b), \ldots, E(\mathtt{pk}_t, b)).$$

From 0.9-correct to almost correct

Correctness

- Use G to generate t independent key pairs $\mathtt{pk} = (\mathtt{pk}_1, \ldots, \mathtt{pk}_t)$, $\mathtt{sk} = (\mathtt{sk}_1, \ldots, \mathtt{sk}_t)$.
- Encrypt using independent randomness:

$$E(\mathtt{pk}, b) = (E(\mathtt{pk}_1, b), \ldots, E(\mathtt{pk}_t, b)).$$

- Use majority after decoding:

$$D(\mathtt{sk}, \mathtt{code}) = \mathrm{maj}(D(\mathtt{sk}_1, \mathtt{code}_1), \ldots, D(\mathtt{sk}_t, \mathtt{code}_t))).$$

Let $X_i =$ success in the i-th try.
By Chernoff's inequality

$$\mathbf{Pr}\left\{\sum X_i \leqslant \frac{t}{2}\right\} = \mathbf{Pr}\left\{\sum X_i \leqslant \left(1 - \frac{4}{9}\right) \cdot \underbrace{0.9t}_{\leqslant \mathbf{E}\sum X_i}\right\} < 2^{-\Omega(t)}.$$

From 0.9-correct to almost correct

Correctness

- Use G to generate t independent key pairs $\mathtt{pk} = (\mathtt{pk}_1, \ldots, \mathtt{pk}_t)$, $\mathtt{sk} = (\mathtt{sk}_1, \ldots, \mathtt{sk}_t)$.
- Encrypt using independent randomness:

$$E(\mathtt{pk}, b) = (E(\mathtt{pk}_1, b), \ldots, E(\mathtt{pk}_t, b)).$$

- Use majority after decoding:

$$D(\mathtt{sk}, \mathtt{code}) = \mathrm{maj}(D(\mathtt{sk}_1, \mathtt{code}_1), \ldots, D(\mathtt{sk}_t, \mathtt{code}_t))).$$

Let $X_i =$ success in the i-th try.
By Chernoff's inequality

$$\mathbf{Pr}\left\{\sum X_i \leqslant \frac{t}{2}\right\} = \mathbf{Pr}\left\{\sum X_i \leqslant \left(1 - \frac{4}{9}\right) \cdot \underbrace{0.9t}_{\leqslant \mathbf{E}\sum X_i}\right\} < 2^{-\Omega(t)}.$$

- Not enough: what happened to our security?

From 0.9-correct to almost correct

Security

If A breaks the new system with prob. $1/2 + \varepsilon(n)$, one can break the old system:

1. We are given $\mathtt{pk}_*$ and $\mathtt{code}_* = E(\mathtt{pk}_*, b)$ for unknown $b \in \{0, 1\}$ and $\mathtt{sk}_*$.

From 0.9-correct to almost correct

Security

If A breaks the new system with prob. $1/2 + \varepsilon(n)$, one can break the old system:

1. We are given $\mathtt{pk}_*$ and $\mathtt{code}_* = E(\mathtt{pk}_*, b)$ for unknown $b \in \{0, 1\}$ and $\mathtt{sk}_*$.
2. Pick random $i_* \in [1\,..\,t]$.
3. Generate keys $(\mathtt{pk}_i, \mathtt{sk}_i) \leftarrow G(1^n)$ for all $i \neq i_*$.
4. Run A on $\mathtt{code} = (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \mathtt{code}_*, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1))$.

If A breaks the new system with prob. $1/2 + \varepsilon(n)$, one can break the old system:

1. We are given $\mathtt{pk}_*$ and $\mathtt{code}_* = E(\mathtt{pk}_*, b)$ for unknown $b \in \{0,1\}$ and $\mathtt{sk}_*$.
2. Pick random $i_* \in [1\,..\,t]$.
3. Generate keys $(\mathtt{pk}_i, \mathtt{sk}_i) \leftarrow G(1^n)$ for all $i \neq i_*$.
4. Run A on $\mathtt{code} = (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \mathtt{code}_*, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1))$.

Consider the distributions

$$\begin{aligned}
\mathtt{code}^{i_*,0} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, 0)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)),\\
\mathtt{code} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, b)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)),\\
\mathtt{code}^{i_*,1} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, 1)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)).
\end{aligned}$$

$$\mathbf{Pr}\{\text{success}\} = \frac{1}{2}\mathbf{Pr}\{\text{success for } 1\} + \frac{1}{2}\mathbf{Pr}\{\text{success for } 0\}$$

$$= \frac{1}{2}\left(\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{\mathbf{1},\mathbf{1}}) = 1\} + \sum_{i_*=\mathbf{2}}^{t}\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{i_*,\mathbf{1}}) = 1\}\right) + \frac{1}{2}\left(\sum_{i_*=\mathbf{1}}^{t-\mathbf{1}}\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{i_*,\mathbf{0}}) = 0\} + \frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{t,\mathbf{0}}) = 0\}\right)$$

If A breaks the new system with prob. $1/2 + \varepsilon(n)$, one can break the old system:

1. We are given $\mathtt{pk}_*$ and $\mathtt{code}_* = E(\mathtt{pk}_*, b)$ for unknown $b \in \{0,1\}$ and $\mathtt{sk}_*$.
2. Pick random $i_* \in [1\,..\,t]$.
3. Generate keys $(\mathtt{pk}_i, \mathtt{sk}_i) \leftarrow G(1^n)$ for all $i \neq i_*$.
4. Run A on $\mathtt{code} = (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \mathtt{code}_*, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1))$.

Consider the distributions

$$
\begin{aligned}
\mathtt{code}^{i_*,\mathbf{0}} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, 0)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)),\\
\mathtt{code} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, b)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)),\\
\mathtt{code}^{i_*,\mathbf{1}} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, 1)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)).\\
\mathtt{code}^{i_*,\mathbf{1}} &= \mathtt{code}^{i_*-1,\mathbf{0}}
\end{aligned}
$$

$$
\begin{aligned}
&\mathbf{Pr}\{\text{success}\} = \frac{1}{2}\mathbf{Pr}\{\text{success for } 1\} + \frac{1}{2}\mathbf{Pr}\{\text{success for } 0\}\\
&= \frac{1}{2}\left(\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{\mathbf{1},\mathbf{1}}) = 1\} + \sum_{i_*=\mathbf{2}}^{t}\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{i_*,\mathbf{1}}) = 1\}\right) + \frac{1}{2}\left(\sum_{i_*=\mathbf{1}}^{t-\mathbf{1}}\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{i_*,\mathbf{0}}) = 0\} + \frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{t,\mathbf{0}}) = 0\}\right)\\
&= \frac{1}{2t}\left(\mathbf{Pr}\{A(\mathtt{code}^{\mathbf{1},\mathbf{1}})) = 1\} + \mathbf{Pr}\{A(\mathtt{code}^{t,\mathbf{0}}) = 0\}\right) + \frac{t-1}{t}\cdot\frac{1}{2}
\end{aligned}
$$

If A breaks the new system with prob. $1/2 + \varepsilon(n)$, one can break the old system:

1. We are given $\mathtt{pk}_*$ and $\mathtt{code}_* = E(\mathtt{pk}_*, b)$ for unknown $b \in \{0,1\}$ and $\mathtt{sk}_*$.
2. Pick random $i_* \in [1\,..\,t]$.
3. Generate keys $(\mathtt{pk}_i, \mathtt{sk}_i) \leftarrow G(1^n)$ for all $i \neq i_*$.
4. Run A on $\mathtt{code} = (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \mathtt{code}_*, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1))$.

Consider the distributions

$$\begin{aligned}
\mathtt{code}^{i_*,0} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, 0)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)),\\
\mathtt{code} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, b)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)),\\
\mathtt{code}^{i_*,1} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, 1)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)).\\
\mathtt{code}^{i_*,1} &= \mathtt{code}^{i_*-1,0}
\end{aligned}$$

$$\mathbf{Pr}\{\text{success}\} = \frac{1}{2}\mathbf{Pr}\{\text{success for } 1\} + \frac{1}{2}\mathbf{Pr}\{\text{success for } 0\}$$

$$= \frac{1}{2}\left(\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{\mathbf{1},\mathbf{1}}) = 1\} + \sum_{i_*=\mathbf{2}}^{t}\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{i_*,\mathbf{1}}) = 1\}\right) + \frac{1}{2}\left(\sum_{i_*=\mathbf{1}}^{t-\mathbf{1}}\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{i_*,\mathbf{0}}) = 0\} + \frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{t,\mathbf{0}}) = 0\}\right)$$

$$= \frac{1}{2t}\left(\mathbf{Pr}\{A(\mathtt{code}^{\mathbf{1},\mathbf{1}})) = 1\} + \mathbf{Pr}\{A(\mathtt{code}^{t,\mathbf{0}}) = 0\}\right) + \frac{t-1}{t}\cdot\frac{1}{2} > \frac{1}{2t}\cdot 2\left(\frac{1}{2} + \varepsilon\right) + \frac{t-1}{t}\cdot\frac{1}{2} = \left(\frac{1}{2t} + \frac{\varepsilon}{t}\right) + \left(\frac{1}{2} - \frac{1}{2t}\right) = \frac{1}{2} + \frac{\varepsilon}{t}.$$

If A breaks the new system with prob. $1/2 + \varepsilon(n)$, one can break the old system:

1. We are given $\mathtt{pk}_*$ and $\mathtt{code}_* = E(\mathtt{pk}_*, b)$ for unknown $b \in \{0,1\}$ and $\mathtt{sk}_*$.
2. Pick random $i_* \in [1\,..\,t]$.
3. Generate keys $(\mathtt{pk}_i, \mathtt{sk}_i) \leftarrow G(1^n)$ for all $i \neq i_*$.
4. Run A on $\mathtt{code} = (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \mathtt{code}_*, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1))$.

Consider the distributions

$$\begin{aligned}
\mathtt{code}^{i_*,0} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, 0)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)),\\
\mathtt{code} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, b)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)),\\
\mathtt{code}^{i_*,1} &= (E(\mathtt{pk}_1, 0), \ldots, E(\mathtt{pk}_{i_*-1}, 0), \boldsymbol{E(\mathtt{pk}_*, 1)}, E(\mathtt{pk}_{i_*+1}, 1), E(\mathtt{pk}_t, 1)).\\
\mathtt{code}^{i_*,1} &= \mathtt{code}^{i_*-1,0}
\end{aligned}$$

$$\mathbf{Pr}\{\text{success}\} = \frac{1}{2}\mathbf{Pr}\{\text{success for } 1\} + \frac{1}{2}\mathbf{Pr}\{\text{success for } 0\}$$

$$= \frac{1}{2}\left(\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{\mathbf{1},\mathbf{1}}) = 1\} + \sum_{i_*=\mathbf{2}}^{t}\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{i_*,\mathbf{1}}) = 1\}\right) + \frac{1}{2}\left(\sum_{i_*=\mathbf{1}}^{t-\mathbf{1}}\frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{i_*,\mathbf{0}}) = 0\} + \frac{1}{t}\mathbf{Pr}\{A(\mathtt{code}^{t,\mathbf{0}}) = 0\}\right)$$

$$= \frac{1}{2t}\left(\mathbf{Pr}\{A(\mathtt{code}^{\mathbf{1},\mathbf{1}})) = 1\} + \mathbf{Pr}\{A(\mathtt{code}^{t,\mathbf{0}}) = 0\}\right) + \frac{t-1}{t}\cdot\frac{1}{2} > \frac{1}{2t}\cdot 2\left(\frac{1}{2} + \varepsilon\right) + \frac{t-1}{t}\cdot\frac{1}{2} = \left(\frac{1}{2t} + \frac{\varepsilon}{t}\right) + \left(\frac{1}{2} - \frac{1}{2t}\right) = \frac{1}{2} + \frac{\varepsilon}{t}.$$

Since it was not possible for any non-negligible $\varepsilon(n)$, it remains so.

Certification

Dropping incorrect systems

- We can assume that a secure 0.9-PKCS (G, E, D) runs in time $O(n^2)$:
  if it requires n^k steps, the system $G'(1^n) = G(1^{n^{1/k}})$ requires $O(n^2)$ steps and is also secure.

Certification

Dropping incorrect systems

- We can assume that a secure 0.9-PKCS (G, E, D) runs in time $O(n^2)$:
  if it requires n^k steps, the system $G'(1^n) = G(1^{n^{1/k}})$ requires $O(n^2)$ steps and is also secure.
- Also assume that a secure (G, E, D) has error at most $\frac{1}{8n}$:
  we reduced error at the cost of a $O(n)$ factor in time.

  Thus interrupt G, E, D if they take more than n^5 steps.
  If a secure 0.9-PKCS exists, then a secure n^5-time $(1 - \frac{1}{8n})$-PKCS exists.
  Exercise: what about a constant? What if for small n we cannot encrypt that fast?

Certification

Dropping incorrect systems

- We can assume that a secure 0.9-PKCS (G, E, D) runs in time $O(n^2)$:
  if it requires n^k steps, the system $G'(1^n) = G(1^{n^{1/k}})$ requires $O(n^2)$ steps and is also secure.
- Also assume that a secure (G, E, D) has error at most $\frac{1}{8n}$:
  we reduced error at the cost of a $O(n)$ factor in time.

  Thus interrupt G, E, D if they take more than n^5 steps.
  If a secure 0.9-PKCS exists, then a secure n^5-time $(1 - \frac{1}{8n})$-PKCS exists.
  Exercise: what about a constant? What if for small n we cannot encrypt that fast?
- Try encoding–decoding n times (with fresh keys).
  If the share of wrong answers $< \frac{1}{6n}$, the system is certified.

- We can assume that a secure 0.9-PKCS (G, E, D) runs in time $O(n^2)$:
  if it requires n^k steps, the system $G'(1^n) = G(1^{n^{1/k}})$ requires $O(n^2)$ steps and is also secure.
- Also assume that a secure (G, E, D) has error at most $\frac{1}{8n}$:
  we reduced error at the cost of a $O(n)$ factor in time.

  Thus interrupt G, E, D if they take more than n^5 steps.
  If a secure 0.9-PKCS exists, then a secure n^5-time $(1 - \frac{1}{8n})$-PKCS exists.
  Exercise: what about a constant? What if for small n we cannot encrypt that fast?
- Try encoding–decoding n times (with fresh keys).
  If the share of wrong answers $< \frac{1}{6n}$, the system is certified.

  Chernoff inequalities give us:
  - a $(1 - \frac{1}{8n})$-PKCS is not certified with probability $2^{-\Omega(n)}$,
  - a non-$(1 - \frac{1}{4n})$-PKCS is certified with probability $2^{-\Omega(n)}$.

A complete PKCS

Assembling everything together

$G^*(1^n)$: *// generate keys for good systems out of the first n*

- Certificate $(G_1, E_1, D_1), \ldots, (G_n, E_n, D_n)$, let $I := \{\text{those certified}\} \subseteq \{1, \ldots, n\}$.
- Output $(\texttt{pk}, \texttt{sk}) = (G_i(1^n))_{i \in I}$. *// Set I is also included in* `pk` *and in* `sk`*, we don't write it for brevity*

A complete PKCS

Assembling everything together

$G^*(1^n)$: *// generate keys for good systems out of the first n*

- Certificate $(G_1, E_1, D_1), \ldots, (G_n, E_n, D_n)$, let $I := \{\text{those certified}\} \subseteq \{1, \ldots, n\}$.
- Output $(\texttt{pk}, \texttt{sk}) = (G_i(1^n))_{i \in I}$. *// Set I is also included in* `pk` *and in* `sk`*, we don't write it for brevity*

$E^*(\texttt{pk}, b)$: *// split $b = \oplus b_i$ and encode each b_i separately*

- For every $i \in I$, $b_i \leftarrow U_1$; the last one gets $b_{|I|} := b \oplus \bigoplus_{i<|I|} b_i$ (also U_1).
- Output $\texttt{code} = (E_i(\texttt{pk}_i, b_i))_{i \in I}$.

A complete PKCS

Assembling everything together

$G^*(1^n)$: *// generate keys for good systems out of the first n*

- Certificate $(G_1, E_1, D_1), \ldots, (G_n, E_n, D_n)$, let $I := \{\text{those certified}\} \subseteq \{1, \ldots, n\}$.
- Output $(\mathtt{pk}, \mathtt{sk}) = (G_i(1^n))_{i \in I}$. *// Set I is also included in $\mathtt{pk}$ and in $\mathtt{sk}$, we don't write it for brevity*

$E^*(\mathtt{pk}, b)$: *// split $b = \oplus b_i$ and encode each b_i separately*

- For every $i \in I$, $b_i \leftarrow U_1$; the last one gets $b_{|I|} := b \oplus \bigoplus_{i<|I|} b_i$ (also U_1).
- Output $\mathtt{code} = (E_i(\mathtt{pk}_i, b_i))_{i \in I}$.

$D^*(\mathtt{sk}, \mathtt{code})$:

- Output $\bigoplus_{i \in I} D_i(\mathtt{sk}_i, \mathtt{code}_i)$.

A complete PKCS

Assembling everything together

$G^*(1^n)$: *// generate keys for good systems out of the first n*

- Certificate $(G_1, E_1, D_1), \ldots, (G_n, E_n, D_n)$, let $I := \{\text{those certified}\} \subseteq \{1, \ldots, n\}$.
- Output $(\mathtt{pk}, \mathtt{sk}) = (G_i(1^n))_{i \in I}$. *// Set I is also included in $\mathtt{pk}$ and in $\mathtt{sk}$, we don't write it for brevity*

$E^*(\mathtt{pk}, b)$: *// split $b = \oplus b_i$ and encode each b_i separately*

- For every $i \in I$, $b_i \leftarrow U_1$; the last one gets $b_{|I|} := b \oplus \bigoplus_{i<|I|} b_i$ (also U_1).
- Output $\mathtt{code} = (E_i(\mathtt{pk}_i, b_i))_{i \in I}$.

$D^*(\mathtt{sk}, \mathtt{code})$:

- Output $\bigoplus_{i \in I} D_i(\mathtt{sk}_i, \mathtt{code}_i)$.

Correctness.

- If (G_i, E_i, D_i) are $(1 - \frac{1}{4n})$-PKCS, then total error $\leqslant \frac{n}{4n} \leqslant \frac{1}{4}$.
- Add certification errors (exponentially small).

A complete PKCS

Assembling everything together

$G^*(1^n)$: *// generate keys for good systems out of the first n*

- Certificate $(G_1, E_1, D_1), \ldots, (G_n, E_n, D_n)$, let $I := \{\text{those certified}\} \subseteq \{1, \ldots, n\}$.
- Output $(\mathtt{pk}, \mathtt{sk}) = (G_i(1^n))_{i \in I}$. *// Set I is also included in $\mathtt{pk}$ and in $\mathtt{sk}$, we don't write it for brevity*

$E^*(\mathtt{pk}, b)$: *// split $b = \oplus b_i$ and encode each b_i separately*

- For every $i \in I$, $b_i \leftarrow U_1$; the last one gets $b_{|I|} := b \oplus \bigoplus_{i<|I|} b_i$ (also U_1).
- Output $\mathtt{code} = (E_i(\mathtt{pk}_i, b_i))_{i \in I}$.

$D^*(\mathtt{sk}, \mathtt{code})$:

- Output $\bigoplus_{i \in I} D_i(\mathtt{sk}_i, \mathtt{code}_i)$.

Correctness.

- If (G_i, E_i, D_i) are $(1 - \frac{1}{4n})$-PKCS, then total error $\leqslant \frac{n}{4n} \leqslant \frac{1}{4}$.
- Add certification errors (exponentially small).

Security.

- One needs to compute all the b_i's (transform it into a formal proof NOW!).
- The probability to distinguish each one of them is negligible.
- Also the chances that a bad system was certified are exponentially small.

Now how does it help us?

Complexity as an obstacle:

- We know L is $\mathbf{P}$.
- Usually it is a constructive proof: an algorithm.
- We know the polynomial and the constants and can assess what we can do.

Now how does it help us?

Complexity as an obstacle:

- We know L is **P**.
- Usually it is a constructive proof: an algorithm.
- We know the polynomial and the constants and can assess what we can do.

Complexity as a resource:

- We know L is **NP**-complete.
- If $\mathbf{P} \neq \mathbf{NP}$, then very adversary fails to solve it faster than any polynomial time.

Now how does it help us?

Complexity as an obstacle:

- We know L is **P**.
- Usually it is a constructive proof: an algorithm.
- We know the polynomial and the constants and can assess what we can do.

Complexity as a resource:

- We know L is **NP**-complete.
- If $\mathbf{P} \neq \mathbf{NP}$, then very adversary fails to solve it faster than any polynomial time.
- Still for every n there may be A_n that decides $x \stackrel{?}{\in} L$ in $C_n \cdot n$ steps for every $x \in \{0,1\}^n$.
  - We know how C_n grows asymptotically, but nothing about it for $n < 10^{100}$.
  - A $10n$-time algorithm for $n < 10^{100}$ does not contradict **NP**-completeness.
- Not to say about the probabilities...

Now how does it help us?

Complexity as an obstacle:

- We know L is **P**.
- Usually it is a constructive proof: an algorithm.
- We know the polynomial and the constants and can assess what we can do.

Complexity as a resource:

- We know L is **NP**-complete.
- If $\mathbf{P} \neq \mathbf{NP}$, then very adversary fails to solve it faster than any polynomial time.
- Still for every n there may be A_n that decides $x \stackrel{?}{\in} L$ in $C_n \cdot n$ steps for every $x \in \{0,1\}^n$.
  - We know how C_n grows asymptotically, but nothing about it for $n < 10^{100}$.
  - A $10n$-time algorithm for $n < 10^{100}$ does not contradict **NP**-completeness.
- Not to say about the probabilities...
- Can we boost our security?

Now how does it help us?

Complexity as an obstacle:

- We know L is **P**.
- Usually it is a constructive proof: an algorithm.
- We know the polynomial and the constants and can assess what we can do.

Complexity as a resource:

- We know L is **NP**-complete.
- If $\mathbf{P} \neq \mathbf{NP}$, then very adversary fails to solve it faster than any polynomial time.
- Still for every n there may be A_n that decides $x \stackrel{?}{\in} L$ in $C_n \cdot n$ steps for every $x \in \{0,1\}^n$.
  - We know how C_n grows asymptotically, but nothing about it for $n < 10^{100}$.
  - A $10n$-time algorithm for $n < 10^{100}$ does not contradict **NP**-completeness.
- Not to say about the probabilities...
- Can we boost our security? Yes.

Now how does it help us?

Complexity as an obstacle:

- We know L is **P**.
- Usually it is a constructive proof: an algorithm.
- We know the polynomial and the constants and can assess what we can do.

Complexity as a resource:

- We know L is **NP**-complete.
- If $\mathbf{P} \neq \mathbf{NP}$, then very adversary fails to solve it faster than any polynomial time.
- Still for every n there may be A_n that decides $x \stackrel{?}{\in} L$ in $C_n \cdot n$ steps for every $x \in \{0,1\}^n$.
  - We know how C_n grows asymptotically, but nothing about it for $n < 10^{100}$.
  - A $10n$-time algorithm for $n < 10^{100}$ does not contradict **NP**-completeness.
- Not to say about the probabilities...
- Can we boost our security? Yes. Does it help?

Now how does it help us?

Complexity as an obstacle:

- We know L is **P**.
- Usually it is a constructive proof: an algorithm.
- We know the polynomial and the constants and can assess what we can do.

Complexity as a resource:

- We know L is **NP**-complete.
- If $\mathbf{P} \neq \mathbf{NP}$, then very adversary fails to solve it faster than any polynomial time.
- Still for every n there may be A_n that decides $x \stackrel{?}{\in} L$ in $C_n \cdot n$ steps for every $x \in \{0,1\}^n$.
  - We know how C_n grows asymptotically, but nothing about it for $n < 10^{100}$.
  - A $10n$-time algorithm for $n < 10^{100}$ does not contradict **NP**-completeness.
- Not to say about the probabilities...
- Can we boost our security? Yes. Does it help? No: time matters.

Now how does it help us?

Complexity as an obstacle:

- We know L is **P**.
- Usually it is a constructive proof: an algorithm.
- We know the polynomial and the constants and can assess what we can do.

Complexity as a resource:

- We know L is **NP**-complete.
- If $\mathbf{P} \neq \mathbf{NP}$, then very adversary fails to solve it faster than any polynomial time.
- Still for every n there may be A_n that decides $x \stackrel{?}{\in} L$ in $C_n \cdot n$ steps for every $x \in \{0,1\}^n$.
  - We know how C_n grows asymptotically, but nothing about it for $n < 10^{100}$.
  - A $10n$-time algorithm for $n < 10^{100}$ does not contradict **NP**-completeness.
- Not to say about the probabilities...
- Can we boost our security? Yes. Does it help? No: time matters.

Solution: quantitative cryptography:

- Make a specific assumption about a primitive: time, key length, probability.
- Inspect reductions (they are constructive): how these parameters change.
- Compute the resulting security parameters.

Takeout

- A complete public-key cryptosystem.
- The unfortunate situation with the grounds of modern cryptography.
- Open ~~question~~ direction: better definitions.
- Practical solution: quantitative cryptography.

Takeout

- A complete public-key cryptosystem.
- The unfortunate situation with the grounds of modern cryptography.
- Open ~~question~~ direction: better definitions.
- Practical solution: quantitative cryptography.

Coming next:

- *(reminder, deadline extended) Homework.*
- *Q-A Session, Exercises and Repeat.*
- *Exam.*

Takeout

- A complete public-key cryptosystem.
- The unfortunate situation with the grounds of modern cryptography.
- Open ~~question~~ direction: better definitions.
- Practical solution: quantitative cryptography.

Coming next:

- *(reminder, deadline extended) Homework.*
- *Q-A Session, Exercises and Repeat.* *Start studying now in order to ask questions.*
- *Exam.*