MTAT.07.003 Cryptology II

How to Model Cryptographic Primitives and Protocols

Sven Laur
University of Tartu

Abstraction is a key to successs

- ▷ **Cryptographic constructions are complex**
  - ◇ Irrelevant techincal details obscure security proofs.
  - ◇ A good abstraction clarifies what is meant by security.
  - ◇ An abstraction highlights which properties are relevant for security.
- ▷ **Cryptographic constructions are not provably secure**
  - ◇ Security of most cryptographic constructions is based on *intractability*.
  - ◇ So far provable lower bounds are *trivial* for all computational problems.
  - ◇ It is also *highly* unlikely that such proofs *do* exist in a *compact* form.
- ▷ **Abstraction allows to escape intractability issues**
  - ◇ We just assume that necessary cryptographic primitives exist.
  - ◇ The actual implementation of such primitives is out of our scope.

Illustrative Example

2048-bit RSA

Key generation

1. Choose two 1024-bit prime numbers p and q.
2. Compute Let $n = pq$, choose $e \xleftarrow[u]{} \mathbb{Z}^*_{\phi(n)}$ and set $d \leftarrow e^{-1} \mod \phi(n)$.
3. Public key is (n, e) and secret key is (n, e, d).

Encryption

1. Let pad : $\{0,1\}^{128} \to \mathbb{Z}^*_n$ be a predefined embedding.
2. To encrypt $m \in \{0,1\}^{128}$, output $c \leftarrow \text{pad}(m)^e \mod n$.

Decryption

1. To decrypt $c \in \mathbb{Z}_n$, compute $x \leftarrow c^d \mod n$.
2. Extract m form x and verify that $\text{pad}(m) = x$.
3. Output $\perp$ in case of failure and m otherwise.

The corresponding abstraction

RSA-2048

Key generation

1. Choose two 1024-bit prime numbers p and q.
2. Compute Let $n = pq$, choose $e \leftarrow \mathbb{Z}^*_{\phi(n)}$ and set $d \leftarrow e^{-1} \mod \phi(n)$.
3. Public key is (n, e) and secret key is (n, e, d).

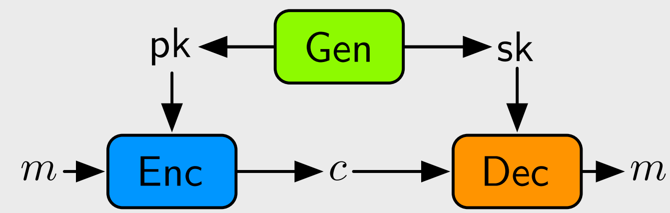
Encryption

1. Let $\mathsf{pad} : \{0,1\}^{128} \to \mathbb{Z}^*_n$ be a predefined embedding.
2. To encrypt $m \in \{0,1\}^{128}$, output $c \leftarrow \mathsf{pad}(m)^e \mod n$.

Decryption

1. To decrypt $c \in \mathbb{Z}_n$, compute $x \leftarrow c^d \mod n$.
2. Extract m form x and verify that $\mathsf{pad}(m) = x$.
3. Output $\perp$ in case of failure and m otherwise.

Public Key Cryptosystem



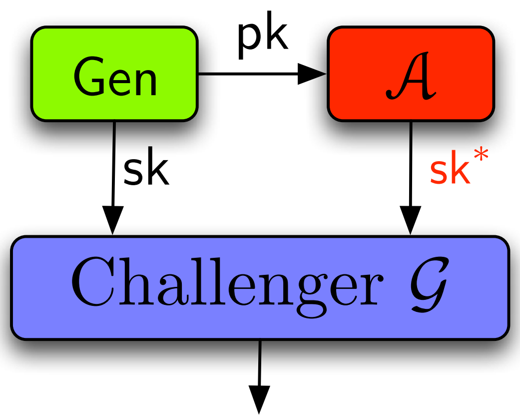
$$\forall(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen} : \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m)) \equiv m$$

To get rid of unnecessary details

- We split the system into algorithms and treat them as black boxes.
- Functionality is guaranteed by specifying additional conditions.
- Security is defined through specifications of tolerable attack scenarios.

Naive security requirement

Goal: It should be infeasible to derive a secret key from accessible data.



$$\mathcal{G}^{\mathcal{A}}$$

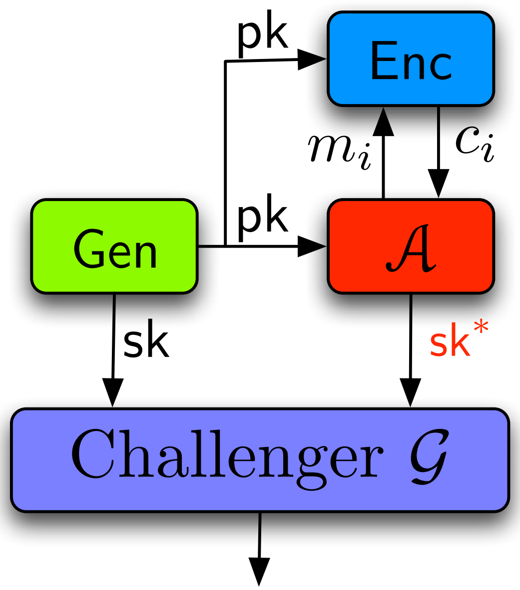
$$\left[\begin{array}{l} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen} \\ \mathsf{sk}^* \leftarrow \mathcal{A}(\mathsf{pk}) \\ \mathbf{return}\ [\mathsf{sk} \stackrel{?}{=} \mathsf{sk}^*] \end{array}\right.$$

The *advantage* of a *key only attack* is defined as an *average* success:

$$\mathsf{Adv}(\mathcal{A}) = \Pr\left[\mathcal{G}^{\mathcal{A}} = 1\right] \quad .$$

Caveat: The attack scenario does not capture the security goal in real life.

Seemingly more advanced attack scenario



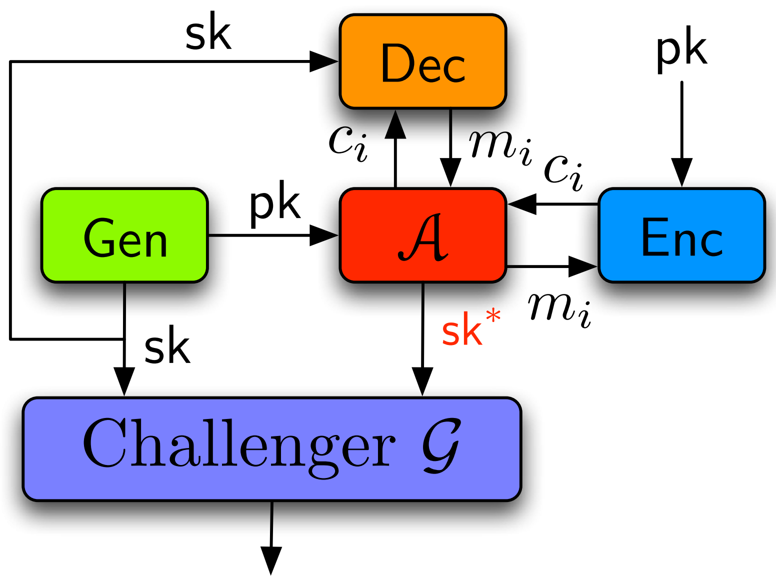
$$\mathcal{G}^{\mathcal{A}}$$

$$\left[\begin{array}{l}(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Gen}\\ \mathsf{sk}^* \leftarrow \mathcal{A}^{\mathsf{Enc}_{\mathsf{pk}}(\cdot)}(\mathsf{pk})\\ \textbf{return } [\mathsf{sk} \stackrel{?}{=} \mathsf{sk}^*]\end{array}\right.$$

Caveat: The attack scenario is not more powerful than the previous.

- ▷ The adversary $\mathcal{A}$ knows what is inside $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ blocks.
- ▷ As adversary knows pk, she can compute $\mathsf{Enc}_{\mathsf{pk}}(m)$ by herself.
- ▷ The oracle access to $\mathsf{Enc}_{\mathsf{pk}}(\cdot)$ function is redundant.

Classical chosen-ciphertext attack scenario



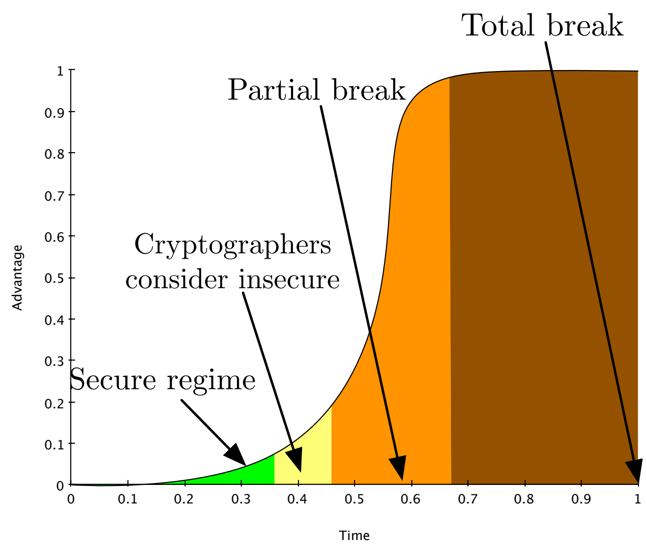
$$\mathcal{G}^{\mathcal{A}}\left[\begin{array}{l}(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Gen}\\ \mathsf{sk}^* \leftarrow \mathcal{A}^{\mathsf{Enc}_{\mathsf{pk}}(\cdot),\mathsf{Dec}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})\\ \textbf{return } [\mathsf{sk} \stackrel{?}{=} \mathsf{sk}^*]\end{array}\right.$$

The difference: The attacker has an implicit access to secret key.

- Decryption operation can leak information about secret key.
- This can happen only for the messages not computed by $\mathsf{Enc}_{\mathsf{pk}}(\cdot)$.
- Such attacks are sometimes plausible in real life.

Time-success profiles

Fix the security game and the advantage function $\mathsf{Adv}(\cdot)$. Then any concrete instantiation of a primitive can be broken with enough resources.



As a result, there exist a time-success profile $\varepsilon = \varepsilon(t)$, which captures the main security properties. Unfortunately, this profile cannot be computed nor approximated with our current knowledge.

Examples of Low-level Primitives

Discrete logarithm

- ▷ Let p be a prime such that $p = 2q + 1$ for another 2048-bit prime q.
- ▷ Fix a generator g such that $g^2 \neq 1$ and define $\mathbb{G} = \left\{g^i : 0 \leq i < q\right\}$.
- ▷ Then discrete logarithm defined below is considered intractable

$$\forall y \in \mathbb{G} : \ \log(y) = x \Leftrightarrow g^x \equiv y \pmod{p} \ .$$

Exercise. Abstract away all details under the assumptions:

- ▷ all construction based on it use only multiplication modulo p,
- ▷ strings are mapped to $\mathbb{G}$ and elements of $\mathbb{G}$ are mapped to strings.

How to model the primitive if constructions also use addition modulo p?

Discrete logarithm problem in an abstract group

$$\{0,1\}^k \longrightarrow \boxed{\mathsf{Emb}} \longrightarrow \boxed{(\mathbb{G};\cdot)} \longrightarrow \boxed{\mathsf{H}} \longrightarrow \{0,1\}^n$$

Definition. Let $\mathbb{G} = \langle g \rangle$ be a q-element multiplicative group generated by the element g. Then for any elements $y, z \in \mathbb{G}$ the discrete logarithm $\log_z y$ is defined as the smallest integer x such that $z^x = y$ and $\perp$ if $y \notin \langle z \rangle$.

Advantage. Let $\mathsf{Adv}^{\mathsf{dl}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[\mathcal{G}^{\mathcal{A}} = 1\right]$ be defined through the game

$$\mathcal{G}^{\mathcal{A}}$$

$$\left[\begin{array}{l} x \xleftarrow[u]{} \mathbb{Z}_q \\ \textbf{return } [x \stackrel{?}{=} \mathcal{A}(g, g^x)] \end{array}\right.$$

Discrete logarithm problem in an abstract group

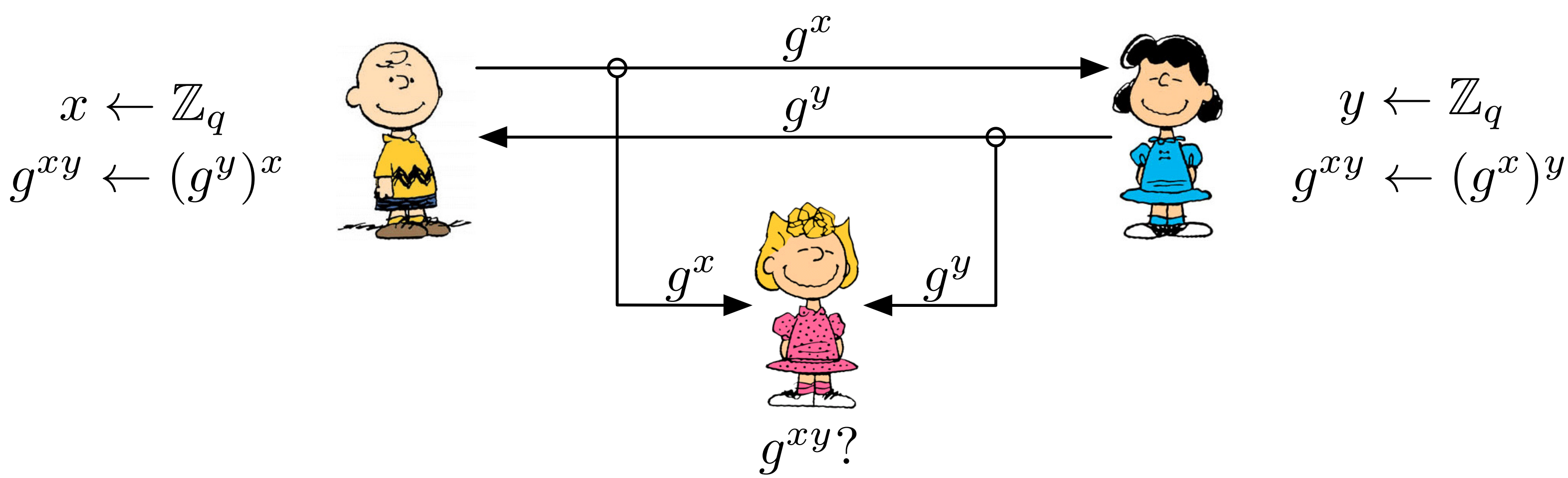
Definition. Let $\mathbb{G} = \langle g \rangle$ be a q-element multiplicative group generated by the element g. Then for any elements $y, z \in \mathbb{G}$ the discrete logarithm $\log_z y$ is defined as the smallest integer x such that $z^x = y$ and $\perp$ if $y \notin \langle z \rangle$.

Advantage. Let $\mathsf{Adv}^{\mathsf{dl}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[\mathcal{G}^{\mathcal{A}} = 1\right]$ be defined through the game

$$\mathcal{G}^{\mathcal{A}}$$
$$\left[\begin{array}{l} x \xleftarrow[u]{} \mathbb{Z}_q \\ \textbf{return } [x \stackrel{?}{=} \mathcal{A}(g, g^x)] \end{array}\right.$$

Security. A group $\mathbb{G}$ is (t, ε)-*secure* DL-group iff for any t-time adversary $\mathcal{A}$ the corresponding advantage $\mathsf{Adv}^{\mathsf{dl}}_{\mathbb{G}}(\mathcal{A}) \leq \varepsilon$.

Diffie-Hellman protocol



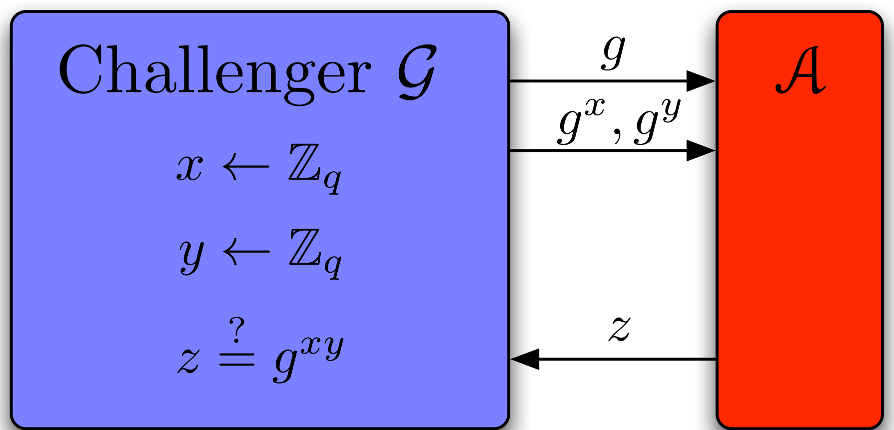
Exercise. Formalise the security requirements for Diffie-Hellman protocol.

1. Eavesdropper cannot reconstruct the common secret g^{xy}.
2. Eavesdropper learns nothing about the common secret g^{xy}.

How to convert the common secret g^{xy} to a valid secret key sk $\in \{0,1\}^n$?

Computational Diffie-Hellman problem

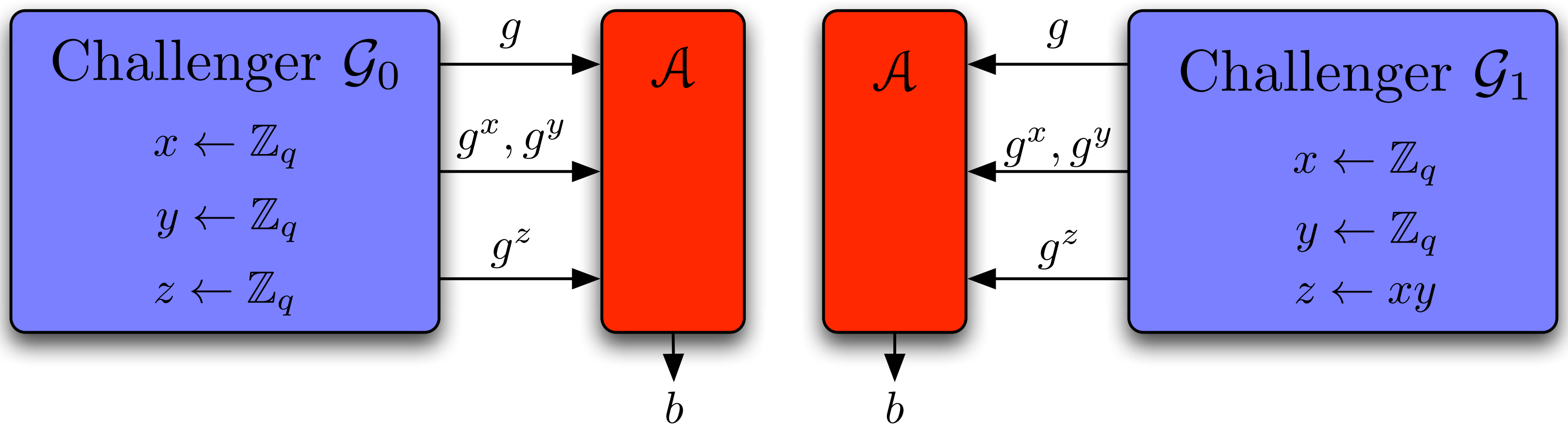
Security. A group $\mathbb{G}$ is (t,ε)*-secure* CDH-group iff for any t-time adversary $\mathcal{A}$ the corresponding advantage $\mathsf{Adv}^{\mathsf{cdh}}_{\mathbb{G}}(\mathcal{A}) \leq \varepsilon$ where the corresponding security game is defined as follows.



$$\mathcal{G}^{\mathcal{A}}\left[\begin{array}{l} x \leftarrow \mathbb{Z}_q \\ y \leftarrow \mathbb{Z}_q \\ z \leftarrow \mathcal{A}(g, g^x, g^y) \\ \mathbf{return}\ [g^{xy} \stackrel{?}{=} z] \end{array}\right.$$

Decisional Diffie-Hellman

Security. A group $\mathbb{G}$ is *(t, ε)-secure* DDH-group iff for any t-time adversary $\mathcal{A}$ the corresponding advantage $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{A}) \leq \varepsilon$ where the corresponding security games $\mathcal{G}_0$ and $\mathcal{G}_1$ and the advantage are defined as follows.



$$\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{A}) = \left|\Pr\left[\mathcal{G}_0^{\mathcal{A}} = 1\right] - \Pr\left[\mathcal{G}_1^{\mathcal{A}} = 1\right]\right|$$

Factorisation

Factorisation of n-bit composite numbers is considered difficult

- ▷ Naive factorisation takes $\Theta(2^{\frac{n}{2}})$ division operations.
- ▷ Pollard ρ algorithm takes $\mathrm{O}(2^{\frac{n}{4}})$ multiplication operations on average.
- ▷ Quadratic sieve takes $\mathrm{O}(2^{c\sqrt{n}})$ multiplication operations on average.
- ▷ Number field sieve takes $\mathrm{O}(2^{c\sqrt[3]{n}})$ multiplication operations on average.

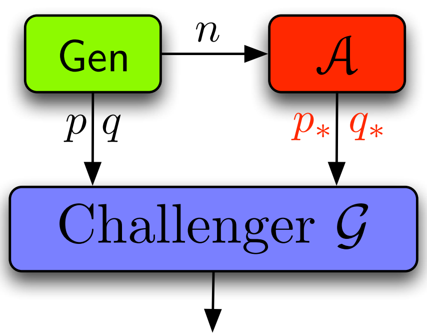
Current records

- ▷ Largest RSA challenge factored had 768 bits (2009).
- ▷ Largest number factored using quantum annealing is 4,088,459 (2018).
- ▷ Largest partially factored Mersenne number has $5,240,707$ bits (2016).
- ▷ Approximate running-times are in thousands of computer years.
- ▷ Shor's algoritm failed to factor 35 on $\mathrm{IBMQX5}$ quantum computer (2019).

Abstract distribution of RSA moduli

Definition. A *distribution of RSA moduli* $\mathfrak{N}$ is defined by an efficient algorithm Gen that outputs n, p, q such that $n = pq$ and p, q are primes.

Security. A distribution $\mathfrak{N}$ is (t, ε)*-secure* RSA-distribution iff for any t-time adversary $\mathcal{A}$ the corresponding advantage $\mathsf{Adv}^{\mathsf{rsa}}_{\mathbb{G}}(\mathcal{A}) \leq \varepsilon$ where the security game is defined as follows



$$\mathcal{G}^{\mathcal{A}}$$

$$\left[\begin{array}{l} (n, p, q) \leftarrow \mathsf{Gen} \\ p_*, q_* \leftarrow \mathcal{A}(n) \\ \textbf{return } [p \stackrel{?}{=} p_*] \vee [p \stackrel{?}{=} q_*] \end{array}\right.$$

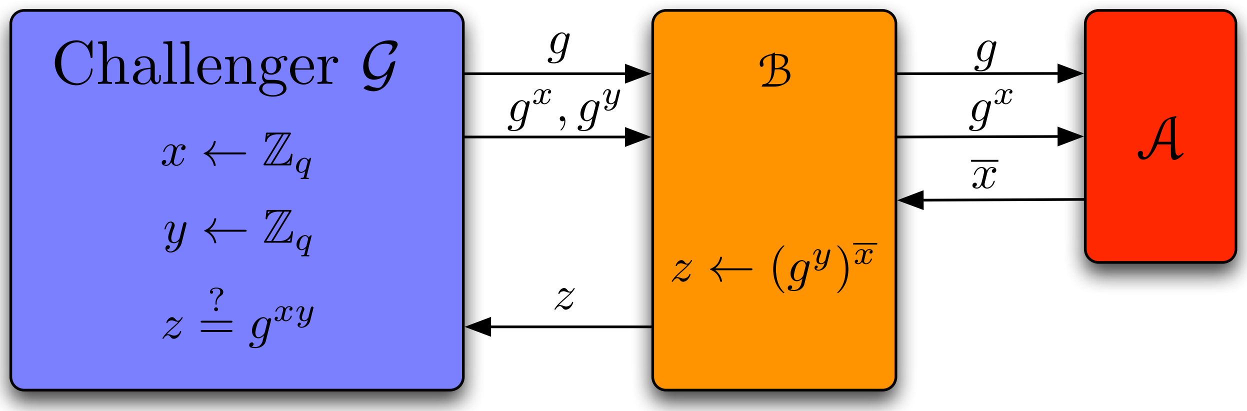
Example. Let $\mathfrak{P}$ be an efficiently samplable set of primes. Then the distribution of products pq where $p \leftarrow \mathfrak{P}$ and $q \leftarrow \mathfrak{P}$ is RSA distribution.

Relations Between Problems

CDH group is also DH group

Intuition: If we can compute discrete logarithm then CDH is easy.

Reduction. Let $\mathcal{A}$ be a DL-finder algorithm. Then the adversary $\mathcal{B}$



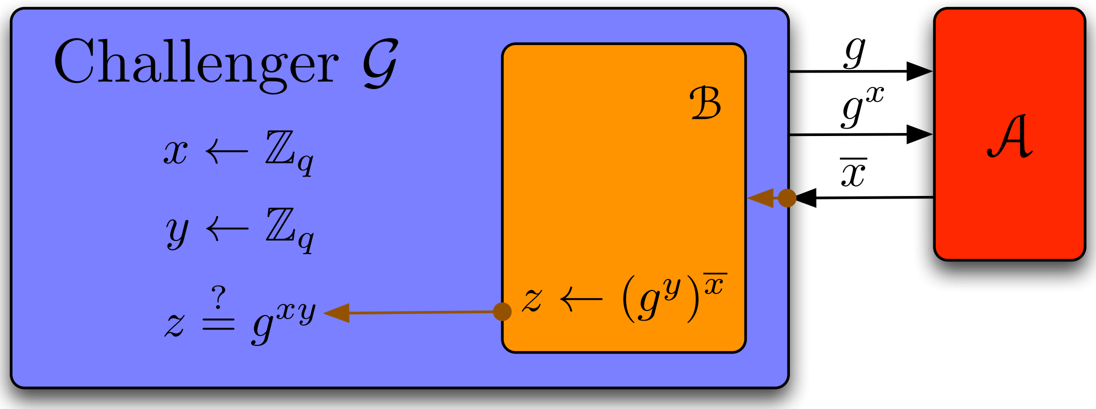
is as successful as the adversary $\mathcal{A}$:

$$\mathsf{Adv}^{\mathsf{cdh}}_{\mathbb{G}}(\mathcal{B}) = \mathsf{Adv}^{\mathsf{dl}}_{\mathbb{G}}(\mathcal{A}) \enspace .$$

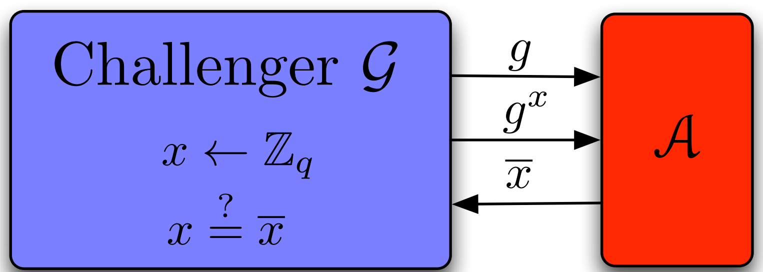
Hence (t, ε)-secure CDH group must be also (t, ε)-secure DL group.

Formal proof

The adversary $\mathcal{A}$ sees the following chain of events

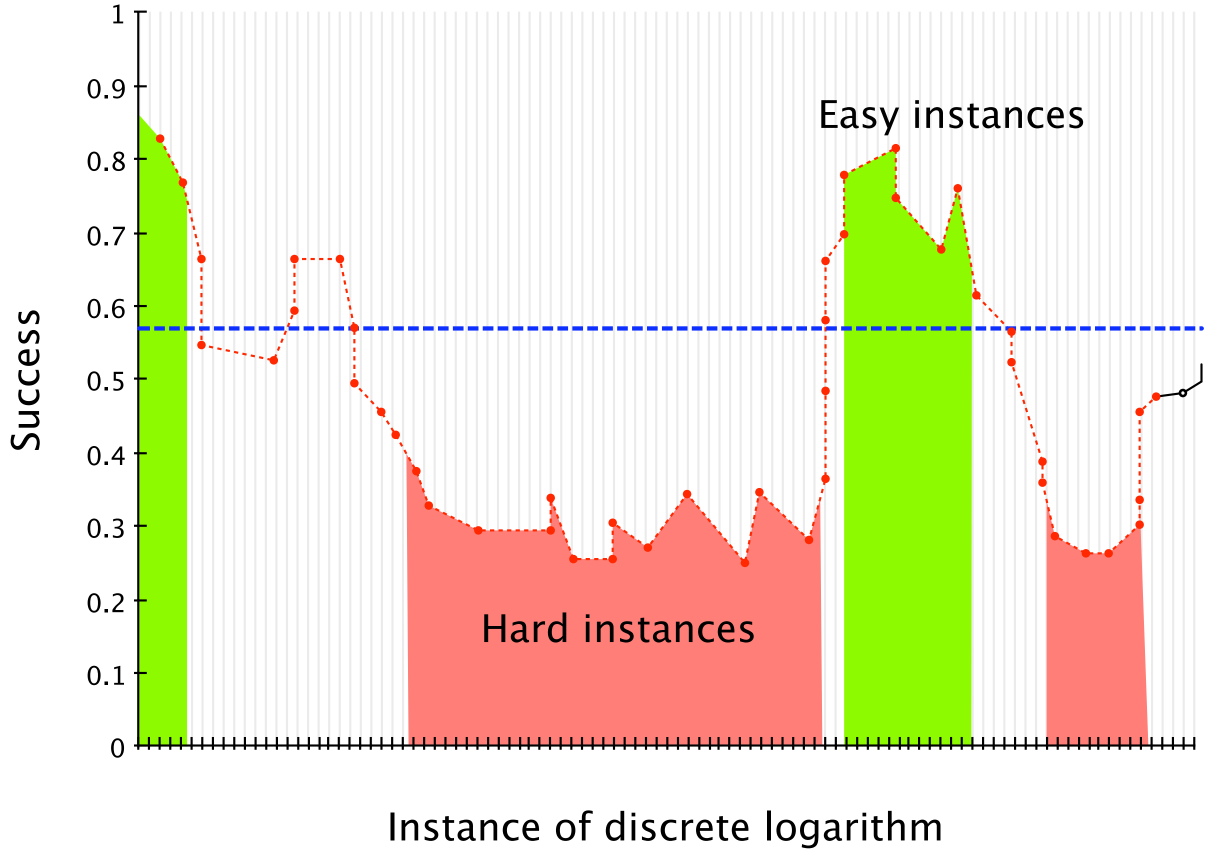


As $z = g^{xy} \Leftrightarrow xy = \overline{x}y \Leftrightarrow x = \overline{x}$ we can further simplify



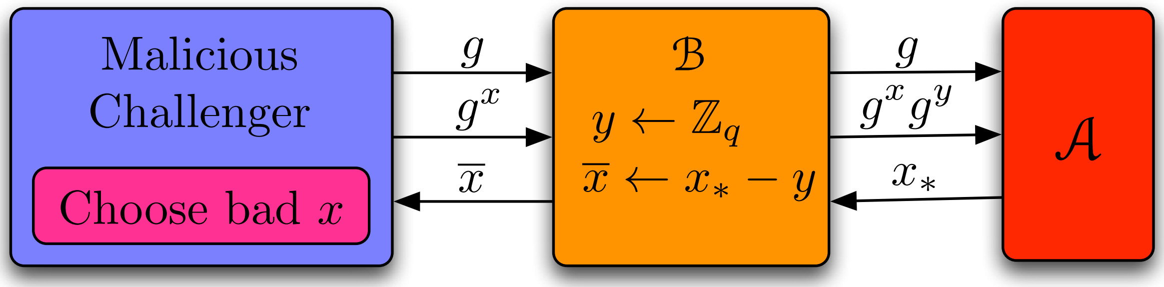
Simple and difficult puzzles

Intuition: A good algorithm *should* work uniformly well on each instance.

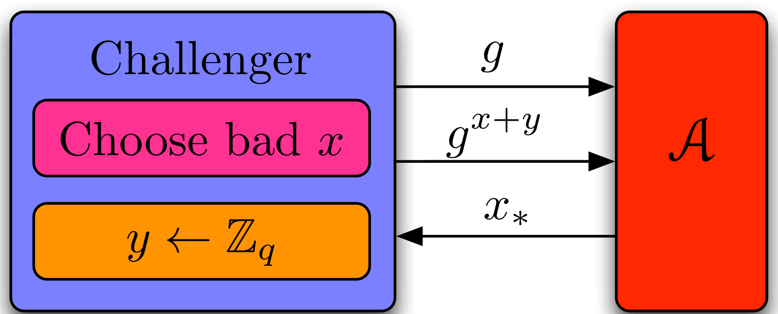


Random self-reducibility

Any instance of a discrete logarithm can be reduced to a random instance.



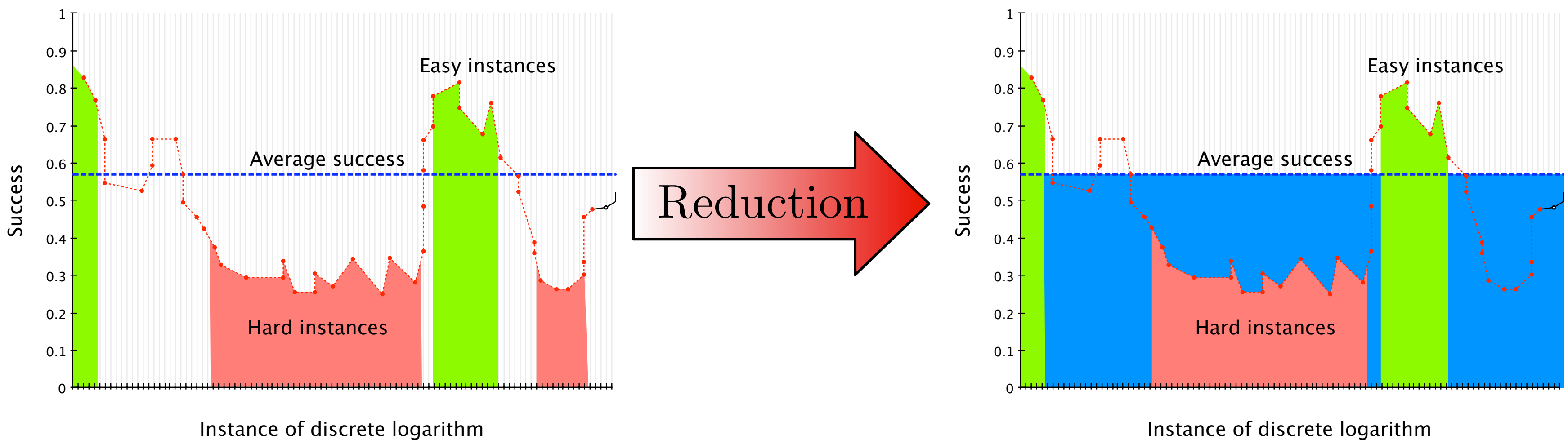
The adversary $\mathcal{A}$ sees the following chain of events



and thus the worst case advantage $\Pr\left[x = \mathcal{B}(g^x)\right] = \mathsf{Adv}^{\mathsf{dl}}_{\mathbb{G}}(\mathcal{A})$.

Consequences of random self-reducibility

Consequence: There are no hard instances but easy instances may exist.



- ▷ The average success is larger for hard instances.
- ▷ Easy instances are handled worse than by the original algorithm.
- ▷ Specialised algorithms for specific instance classes might work better.

Consequences of random self-reducibility

Consequence: There are various trade-offs between time and success.

▷ By repeating the DL-computations we can increase the success.

▷ Any estimate on parameters t, ε gives a lower bound to success.