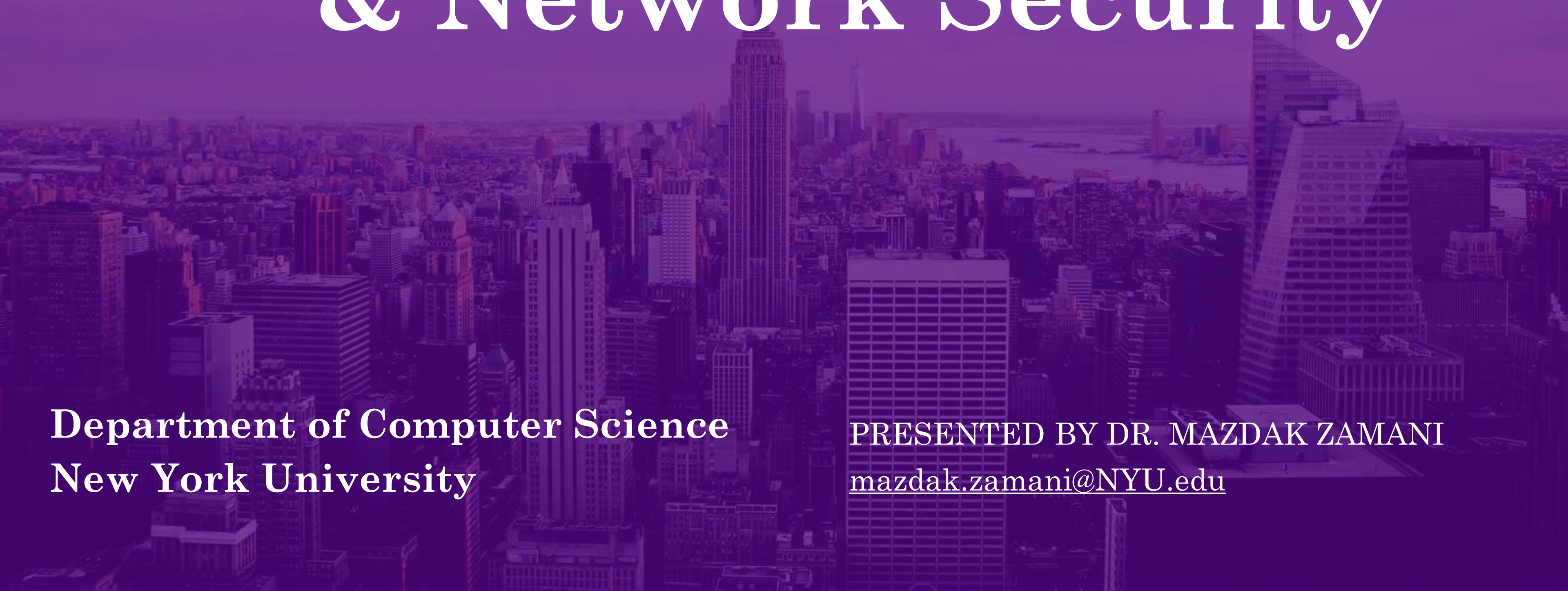


CSCI-GA.3205 Applied Cryptography & Network Security

Department of Computer Science
New York University

PRESENTED BY DR. MAZDAK ZAMANI
mazdak.zamani@NYU.edu

Number Theory
Trapdoor permutation scheme
RSA
Diffie-Hellman key exchange
Merkle puzzles

05

NYU

Basic number theory

- From here on: N denotes a positive integer; p denote a prime.
- Notation: $\mathbb{Z}_{12} = \{ 0, 1, \ldots , 11\}$
    - $9 + 8 = 5$ in $\mathbb{Z}_{12}$
    - $5 \times 7 = 11$ in $\mathbb{Z}_{12}$
    - $5 - 7 = 10$ in $\mathbb{Z}_{12}$
- Def: $\mathbb{Z}_N{}^*$ =(set of invertible elements in $\mathbb{Z}_N$) $= \{ x \in \mathbb{Z}_N : \gcd(x,N) = 1 \}$
    - $\mathbb{Z}_{12}{}^* = \{ 1, 5, 7, 11\}$

Basic number theory

1. For a prime $p > 2$ let $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$.
   Elements of $\mathbb{Z}_p$ can be added modulo p and multiplied modulo p. For $x, y \in \mathbb{Z}_p$ we write $x + y$ and $x \cdot y$ to denote the sum and product of x and y modulo p.

2. Fermat's theorem: $g^{p-1} = 1$ for all $0 \neq g \in \mathbb{Z}_p$
   Example: $3^4 = 81 \equiv 1 \pmod 5$.

3. The *inverse* of $x \in \mathbb{Z}_p$ is an element $a \in \mathbb{Z}_p$ satisfying $a \cdot x = 1$ in $\mathbb{Z}_p$.
   The inverse of x in $\mathbb{Z}_p$ is denoted by x^{-1}.
   Example: 1. 3^{-1} in $\mathbb{Z}_5$ is 2 since $2 \cdot 3 \equiv 1 \pmod 5$.
   2. 2^{-1} in $\mathbb{Z}_p$ is $\frac{p+1}{2}$.

NYU

Basic number theory

4. All elements $x \in \mathbb{Z}_p$ except for $x = 0$ are invertible.
   Simple (but inefficient) inversion algorithm: $x^{-1} = x^{p-2}$ in $\mathbb{Z}_p$.
   Indeed, $x^{p-2} \cdot x = x^{p-1} = 1$ in $\mathbb{Z}_p$.

5. We denote by $\mathbb{Z}_p^*$ the set of invertible elements in $\mathbb{Z}_p$. Then $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$.

6. We now have algorithm for solving linear equations in $\mathbb{Z}_p$: $a \cdot x = b$.
   Solution: $x = b \cdot a^{-1} = b \cdot a^{p-2}$.

NYU

Basic number theory

A.1 Cyclic groups

Notation: for a finite cyclic group $\mathbb{G}$ we let $\mathbb{G}^*$ denote the set of generators of $\mathbb{G}$.

A.2 Arithmetic modulo primes

A.2.1 Basic concepts

We use the letters p and q to denote prime numbers. We will be using large primes, e.g. on the order of 300 digits (1024 bits).

Basic number theory

A.2.2 Structure of $\mathbb{Z}_p^*$

1. $\mathbb{Z}_p^*$ is a *cyclic group*.
   In other words, there exists $g \in \mathbb{Z}_p^*$ such that $\mathbb{Z}_p^* = \{1, g, g^2, g^3, \ldots, g^{p-2}\}$.
   Such a g is called a *generator* of $\mathbb{Z}_p^*$.
   Example: in $\mathbb{Z}_7^*$: $\langle 3 \rangle = \{1, 3, 3^2, 3^3, 3^4, 3^5 \boxed{\;}\} \equiv \{1, 3, 2, 6, 4, 5\} \pmod 7 = \mathbb{Z}_7^*$.

2. Not every element of $\mathbb{Z}_p^*$ is a generator.
   Example: in $\mathbb{Z}_7^*$ we have $\langle 2 \rangle = \{1, 2, 4\} \neq \mathbb{Z}_7^*$.

Basic number theory

3. The *order* of $g \in \mathbb{Z}_p^*$ is the smallest positive integer a such that $g^a = 1$.
   The order of $g \in \mathbb{Z}_p^*$ is denoted $\text{order}_p(g)$.
   Example: $\text{order}_7(3) = 6$ and $\text{order}_7(2) = 3$.

4. Lagrange's theorem: for all $g \in \mathbb{Z}_p^*$ we have that $\text{order}_p(g)$ divides $p-1$. Observe that Fermat's theorem is a simple corollary:
   $$\text{for } g \in \mathbb{Z}_p^* \text{ we have } g^{p-1} = (g^{\text{order}(g)})^{(p-1)/\text{order}(g)} = (1)^{(p-1)/\text{order}(g)} = 1.$$

5. If the factorization of $p-1$ is known then there is a simple and efficient algorithm to determine $\text{order}_p(g)$ for any $g \in \mathbb{Z}_p^*$.

- Generalization of Fermat is the basis of the RSA cryptosystem.

Basic number theory

<u>Def</u>: For an integer N define $\phi(N) = |(Z_N)^*|$ (Euler's ϕ func.)

$\phi(12) = |\{1,5,7,11\}| = 4$;

$\phi(p) = p-1$

For $N=p\cdot q$: $\phi(N) = N-p-q+1 = (p-1)(q-1)$

<u>Thm</u> (Euler): $\forall\, x \in (Z_N)^*$: $x^{\phi(N)} = 1$ **in** Z_N

$5^{\phi(12)} = 5^4 = 625 = 1$ in Z_{12}

Key exchange

- Two users, Alice and Bob, who never met before talk on the phone. They are worried that an eavesdropper is listening to their conversation and hence they wish to encrypt the session.
- Since Alice and Bob never met before they have no shared secret key with which to encrypt the session. Thus, their initial goal is to generate a shared secret unknown to the adversary. They may later use this secret as a session-key for secure communication.

Anonymous key exchange

- To do so, Alice and Bob execute a protocol where they take turns in sending messages to each other.
- The eavesdropping adversary can hear all these messages but cannot change them or inject his own messages.
- At the end of the protocol Alice and Bob should have a secret that is unknown to the adversary.
- The protocol itself provides no assurance to Alice that she is really talking to Bob, and no assurance to Bob that he is talking to Alice -- in this sense, the protocol is “anonymous.”

Transcript of protocol (T_P)

- The transcript T_P of protocol P is a random variable, which is a function of the random bits generated by A and B.
- The eavesdropping adversary ***A*** sees the entire transcript T_P and its goal is to figure out the secret k.
- We define security of a key exchange protocol using the following game.

NYU

Attack Game (Anonymous key exchange)

Attack Game 10.1 (Anonymous key exchange). For a key exchange protocol $P = (A, B)$ and a given adversary $\mathcal{A}$, the attack game runs as follows.

- The challenger runs the protocol between A and B to generate a shared key k and transcript T_P. It gives T_P to $\mathcal{A}$.
- $\mathcal{A}$ outputs a guess $\hat{k}$ for k.

We define $\mathcal{A}$'s advantage, denoted $\text{AnonKE}\mathsf{adv}[\mathcal{A}, P]$, as the probability that $\hat{k} = k$. $\square$

Definition 10.1. *We say that an anonymous key exchange protocol P is secure against an eavesdropper if for all efficient adversaries $\mathcal{A}$, the quantity $\text{AnonKE}\mathsf{adv}[\mathcal{A}, P]$ is negligible.*

One-way trapdoor functions

- We introduce a tool that will allow us to build an efficient and secure key exchange protocol.
- One-way function is a function $F : X \rightarrow Y$ that is easy to compute, but hard to invert.
- One-way functions, however, are not sufficient for our purposes. We need one-way functions with a special feature, called a *trapdoor*.
- A *trapdoor* is a secret that allows one to efficiently invert the function; however, without knowledge of the trapdoor, the function remains hard to invert.

NYU

One-way trapdoor functions

Definition 10.2 (Trapdoor function scheme). *Let $\mathcal{X}$ and $\mathcal{Y}$ be finite sets. A* ***trapdoor function scheme*** *$\mathcal{T}$, defined over $(\mathcal{X}, \mathcal{Y})$, is a triple of algorithms (G, F, I), where*

- *G is a probabilistic key generation algorithm that is invoked as $(pk, sk) \xleftarrow{R} G()$, where pk is called a* ***public key*** *and sk is called a* ***secret key****.*

- *F is a deterministic algorithm that is invoked as $y \leftarrow F(pk, x)$, where pk is a public key (as output by G) and x lies in $\mathcal{X}$. The output y is an element of $\mathcal{Y}$.*

- *I is a deterministic algorithm that is invoked as $x \leftarrow I(sk, y)$, where sk is a secret key (as output by G) and y lies in $\mathcal{Y}$. The output x is an element of $\mathcal{X}$.*

Moreover, the following ***correctness property*** *should be satisfied: for all possible outputs (pk, sk) of $G()$, and for all $x \in \mathcal{X}$, we have $I(sk,\ F(pk, x)\ ) = x$.*

Trapdoor permutation scheme

Observe that for every pk, the function $F(pk, \cdot)$ is a function from $\mathcal{X}$ to $\mathcal{Y}$. The correctness property says that sk is the trapdoor for inverting this function; note that this property also implies that the function $F(pk, \cdot)$ is one-to-one. Note that we do not insist that $F(pk, \cdot)$ maps $\mathcal{X}$ onto $\mathcal{Y}$. That is, there may be elements $y \in \mathcal{Y}$ that do not have any preimage under $F(pk, \cdot)$. For such y, we make no requirements on algorithm I — it can return some arbitrary element $x \in \mathcal{X}$ (one might consider returning a special reject symbol in this case, but it simplifies things a bit not to do this).

In the special case where $\mathcal{X} = \mathcal{Y}$, then $F(pk, \cdot)$ is not only one-to-one, but onto. That is, $F(pk, \cdot)$ is a *permutation on the set* $\mathcal{X}$. In this case, we may refer to (G, F, I) as a **trapdoor permutation scheme** defined over $\mathcal{X}$.

The basic security property we want from a trapdoor permutation scheme is a one-wayness property, which basically says that given pk and $F(pk, x)$ for random $x \in \mathcal{X}$, it is hard to compute x without knowledge of the trapdoor sk. This is formalized in the following game.

NYU

Attack Game (One-way trapdoor function scheme)

Attack Game 10.2 (One-way trapdoor function scheme). For a given trapdoor function scheme $\mathcal{T} = (G, F, I)$, defined over $(\mathcal{X}, \mathcal{Y})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger computes

$$(pk, sk) \xleftarrow{R} G(), \quad x \xleftarrow{R} \mathcal{X}, \quad y \leftarrow F(pk, x)$$

  and sends (pk, y) to the adversary.

- The adversary outputs $\hat{x} \in \mathcal{X}$.

We define the adversary's advantage in inverting $\mathcal{T}$, denoted $\mathsf{OWadv}[\mathcal{A}, \mathcal{T}]$, to be the probability that $\hat{x} = x$. $\square$

Definition 10.3. *We say that a trapdoor function scheme $\mathcal{T}$ is* ***one way*** *if for all efficient adversaries $\mathcal{A}$, the quantity $\mathsf{OWadv}[\mathcal{A}, \mathcal{T}]$ is negligible.*

Key exchange using a one-way trapdoor function scheme

We now show how to use a one-way trapdoor function scheme $\mathcal{T} = (G, F, I)$, defined over $(\mathcal{X}, \mathcal{Y})$, to build a secure anonymous key exchange protocol. The protocol runs as follows, as shown in Fig. 10.1:

- Alice computes $(pk, sk) \overset{\mathrm{R}}{\leftarrow} G()$, and sends pk to Bob.
- Upon receiving pk from Alice, Bob computes $x \overset{\mathrm{R}}{\leftarrow} \mathcal{X}, y \leftarrow F(pk, x)$, and sends y to Alice.
- Upon receiving y from Bob, Alice computes $x \leftarrow I(sk, y)$.

Key exchange using a trapdoor function scheme

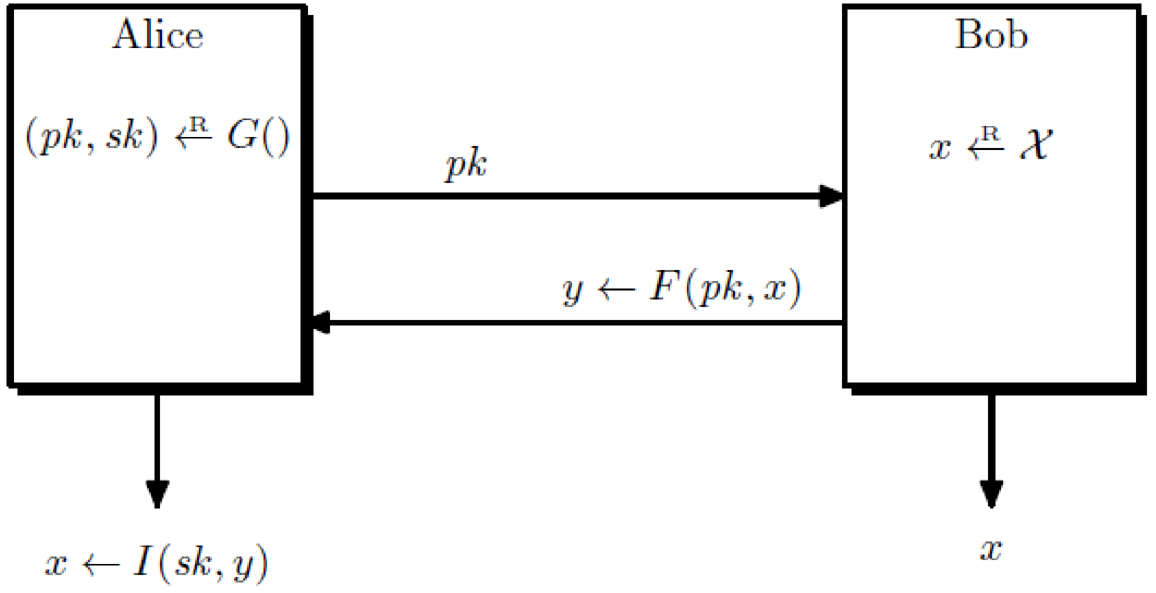


Figure 10.1: Key exchange using a trapdoor function scheme

A trapdoor permutation scheme based on RSA

- We now describe a trapdoor permutation scheme that is plausibly one-way. It is called ***RSA*** after its inventors, Rivest, Shamir, and Adleman.
- Recall that a trapdoor permutation is a special case of a trapdoor function, where the domain and range are the same set. This means that for every public-key, the function is a permutation of its domain, which is why we call it a trapdoor permutation.
- Despite many years of study, RSA is essentially the only known reasonable candidate trapdoor permutation scheme (there are a few others, but they are all very closely related to the RSA scheme).

How RSA works

Here is how RSA works. First, we describe a probabilistic algorithm RSAGen that takes as input an integer $\ell > 2$, and an odd integer $e > 2$.

```
RSAGen($\ell, e$) :=
    generate a random $\ell$-bit prime $p$ such that $\gcd(e, p-1) = 1$
    generate a random $\ell$-bit prime $q$ such that $\gcd(e, q-1) = 1$ and $q \neq p$
    $n \leftarrow pq$
    $d \leftarrow e^{-1} \bmod (p-1)(q-1)$
    output $(n, d)$.
```

To efficiently implement the above algorithm, we need an efficient algorithm to generate random ℓ-bit primes. This is discussed in Appendix A. Also, we use the extended Euclidean algorithm (Appendix A) to compute $e^{-1} \bmod (p-1)(q-1)$. Note that since $\gcd(e, p-1) = \gcd(e, q-1) = 1$, it follows that $\gcd(e, (p-1)(q-1)) = 1$, and hence e has a multiplicative inverse modulo $(p-1)(q-1)$.

NYU

RSA trapdoor permutation scheme

Now we describe the RSA trapdoor permutation scheme $\mathcal{T}_{\text{RSA}} = (G, F, I)$. It is parameterized by fixed values of ℓ and e.

- Key generation runs as follows:

$$\begin{aligned} G() := \quad & (n, d) \xleftarrow{R} \text{RSAGen}(\ell, e), \quad pk \leftarrow (n, e), \quad sk \leftarrow (n, d) \\ & \text{output } (pk, sk). \end{aligned}$$

- For a given public key $pk = (n, e)$, and $x \in \mathbb{Z}_n$, we define $F(pk, x) := x^e \in \mathbb{Z}_n$.
- For a given secret key $sk = (n, d)$, and $y \in \mathbb{Z}_n$, we define $I(sk, y) := y^d \in \mathbb{Z}_n$.

Note that although the encryption exponent e is considered to be a fixed system parameter, we also include it as part of the public key pk.

RSA trapdoor permutation scheme

So now we know that $\mathcal{T}_{\text{RSA}}$ satisfies the correctness property of a trapdoor permutation scheme. However, it is not clear that it is one-way. For $\mathcal{T}_{\text{RSA}}$, one-wayness means that there is no efficient algorithm that given n and x^e, where $x \in \mathbb{Z}_n$ is chosen at random, can effectively compute x. It is clear that if $\mathcal{T}_{\text{RSA}}$ is one-way, then it must be hard to factor n; indeed, if it were easy to factor n, then one could compute d in exactly the same way as is done in algorithm RSAGen, and then use d to compute $x = y^d$.

RSA trapdoor permutation scheme

It is widely believed that factoring n is hard, provided ℓ is sufficiently large — typically, ℓ is chosen to be between 1000 and 1500. Moreover, the only known efficient algorithm to invert $\mathcal{T}_{\text{RSA}}$ is to first factor n and then compute d as above. However, there is no known *proof* that the assumption that factoring n is hard implies that $\mathcal{T}_{\text{RSA}}$ is one-way. Nevertheless, based on current evidence, it seems reasonable to conjecture that $\mathcal{T}_{\text{RSA}}$ is indeed one-way. We state this conjecture now as an explicit assumption. As usual, this is done using an attack game.

NYU

Attack Game (RSA)

Attack Game 10.3 (RSA). For given integers $\ell > 2$ and odd $e > 2$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger computes

$$(n, d) \xleftarrow{R} \text{RSAGen}(\ell, e), \quad x \xleftarrow{R} \mathbb{Z}_n, \quad y \leftarrow x^e \in \mathbb{Z}_n$$

  and gives the input (n, y) to the adversary.

- The adversary outputs $\hat{x} \in \mathbb{Z}_n$.

We define the adversary's advantage in breaking RSA, denoted $\mathsf{RSAadv}[\mathcal{A}, \ell, e]$, as the probability that $\hat{x} = x$. $\square$

Definition 10.5 (RSA assumption). *We say that the RSA assumption holds for (ℓ, e) if for all efficient adversaries $\mathcal{A}$, the quantity $\mathsf{RSAadv}[\mathcal{A}, \ell, e]$ is negligible.*

NYU

Key exchange based on the RSA assumption

Consider now what happens when we instantiate the key exchange protocol in Section 10.2.1 with $\mathcal{T}_{\mathrm{RSA}}$. The protocol runs as follows:

- Alice computes $(n, d) \xleftarrow{\mathrm{R}} \mathrm{RSAGen}(\ell, e)$, and sends (n, e) to Bob.
- Upon receiving (n, e) from Alice, Bob computes $x \xleftarrow{\mathrm{R}} \mathbb{Z}_n$, $y \leftarrow x^e$, and sends y to Alice.
- Upon receiving y from Bob, Alice computes $x \leftarrow y^d$.

The secret shared by Alice and Bob is x. The message flow is the same as in Fig. 10.1. Under the RSA assumption, this is a secure anonymous key exchange protocol.

Theorem 10.1. *Let $n = pq$ where p and q are distinct primes. Let e and d be integers such that $ed \equiv 1 \pmod{(p-1)(q-1)}$. Then for all $x \in \mathbb{Z}$, we have $x^{ed} \equiv x \pmod{n}$.*

Proof. The hypothesis that $ed \equiv 1 \pmod{(p-1)(q-1)}$ just means that $ed = 1 + k(p-1)(q-1)$ for some integer k. Certainly, if $x \equiv 0 \pmod{p}$, then $x^{ed} \equiv 0 \equiv x \pmod{p}$; otherwise, if $x \not\equiv 0 \pmod{p}$, then by Fermat's little theorem (Appendix A), we have

$$x^{p-1} \equiv 1 \pmod{p},$$

and so

$$x^{ed} \equiv x^{1+k(p-1)(q-1)} \equiv x \cdot \left(x^{(p-1)}\right)^{k(q-1)} \equiv x \cdot 1^{k(q-1)} \equiv x \pmod{p}.$$

Therefore,

$$x^{ed} \equiv x \pmod{p}.$$

By a symmetric argument, we have

$$x^{ed} \equiv x \pmod{q}.$$

Thus, $x^{ed} - x$ is divisible by the distinct primes p and q, and must therefore be divisible by their product n, which means

$$x^{ed} \equiv x \pmod{n}. \quad \square$$

NYU

Diffie-Hellman key exchange

- In this section, we explore another approach to constructing secure key exchange protocols, which was invented by Diffie and Hellman. Just as with the protocol based on RSA, this protocol will require a bit of algebra and number theory.

Diffie-Hellman key exchange

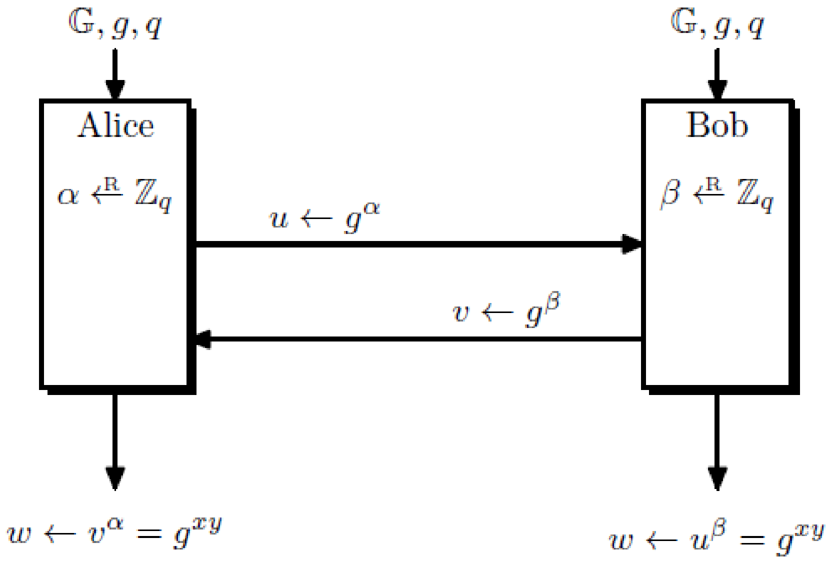
- Alice chooses a random secret α, computes $E(\alpha)$, and sends $E(\alpha)$ to Bob over an insecure channel.
- Likewise, Bob chooses a random secret β, computes $E(\beta)$, and sends $E(\beta)$ to Alice over an insecure channel.
- Alice and Bob both somehow compute a shared key $F(\alpha; \beta)$.
- In this high-level description, E and F are some functions that should satisfy the following properties:
    - E should be easy to compute;
    - given α and $E(\beta)$, it should be easy to compute $F(\alpha; \beta)$;
    - given $E(\alpha)$ and β, it should be easy to compute $F(\alpha; \beta)$;
    - given $E(\alpha)$ and $E(\beta)$, it should be hard to compute $F(\alpha; \beta)$.

Diffie-Hellman key exchange protocol

1. Alice computes $\alpha \xleftarrow{R} \mathbb{Z}_q$, $u \leftarrow g^{\alpha}$, and sends u to Bob.
2. Bob computes $\beta \xleftarrow{R} \mathbb{Z}_q, v \leftarrow g^{\beta}$ and sends v to Alice.
3. Upon receiving v from Bob, Alice computes $w \leftarrow v^{\alpha}$
4. Upon receiving u from Alice, Bob computes $w \leftarrow u^{\beta}$

The secret shared by Alice and Bob is

$$w = v^{\alpha} = g^{\alpha\beta} = u^{\beta}.$$


$\mathbb{G}, g, q$
$\mathbb{G}, g, q$
Alice
$\alpha \xleftarrow{R} \mathbb{Z}_q$
Bob
$\beta \xleftarrow{R} \mathbb{Z}_q$
$u \leftarrow g^{\alpha}$
$v \leftarrow g^{\beta}$
$w \leftarrow v^{\alpha} = g^{xy}$
$w \leftarrow u^{\beta} = g^{xy}$

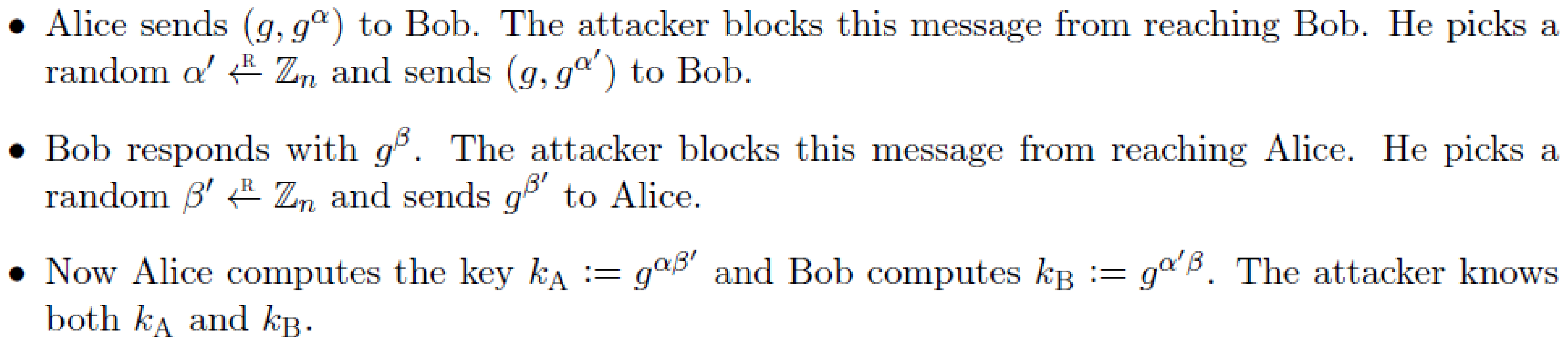
Diffie-Hellman key exchange

To a first approximation, the basic idea is to implement E in terms of exponentiation to some fixed base g, defining $E(\alpha) := g^\alpha$ and $F(\alpha, \beta) := g^{\alpha\beta}$. Notice then that

$$E(\alpha)^\beta = (g^\alpha)^\beta = F(\alpha, \beta) = (g^\beta)^\alpha = E(\beta)^\alpha.$$

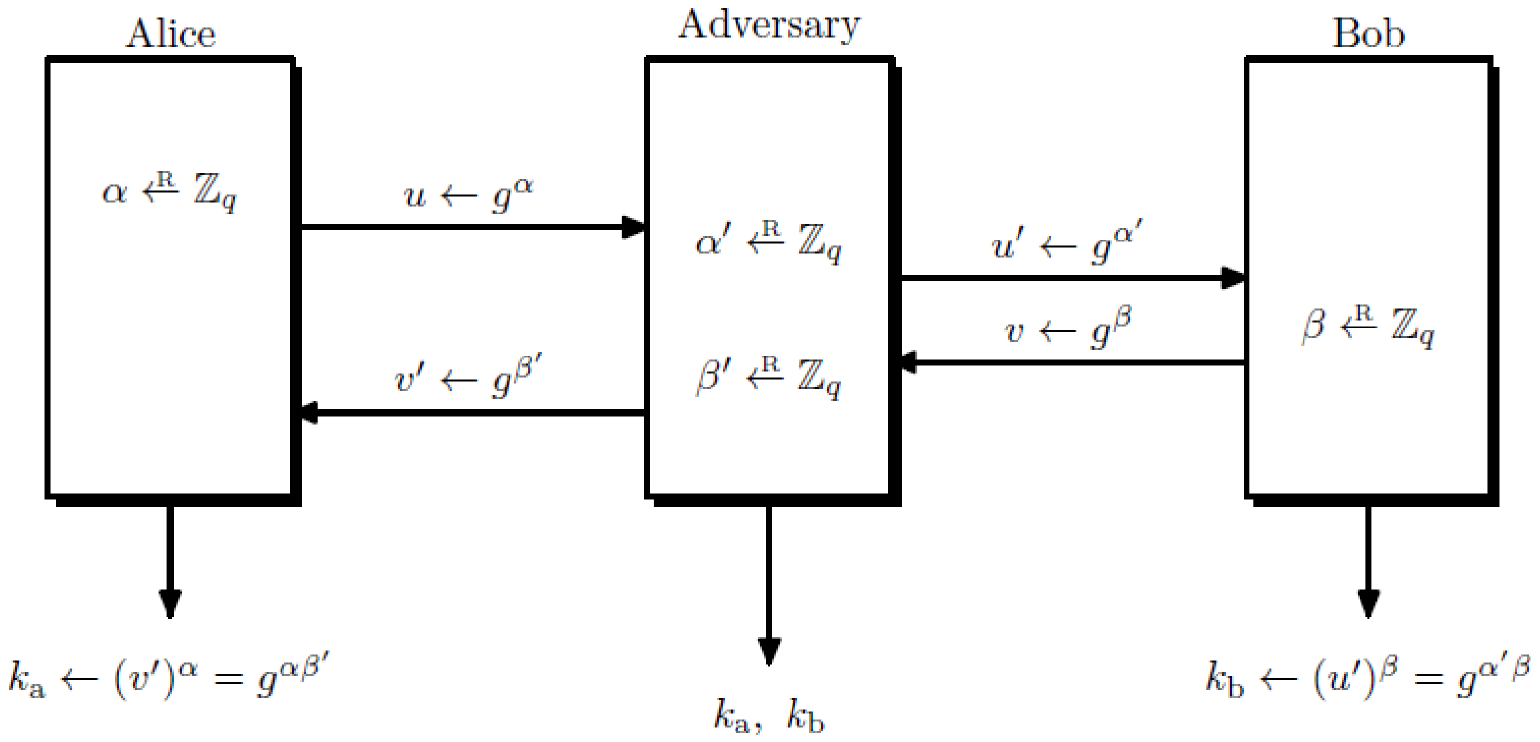
Attacks on the anonymous Diffie-Hellman protocol

- An active attacker can abuse this to expose all traffic between Alice and Bob. The attack, called a **man in the middle attack**, works against any key exchange protocol that does not include authentication. It works as follows:



- Alice sends (g, g^{α}) to Bob. The attacker blocks this message from reaching Bob. He picks a random $\alpha' \xleftarrow{R} \mathbb{Z}_n$ and sends $(g, g^{\alpha'})$ to Bob.
- Bob responds with g^{β}. The attacker blocks this message from reaching Alice. He picks a random $\beta' \xleftarrow{R} \mathbb{Z}_n$ and sends $g^{\beta'}$ to Alice.
- Now Alice computes the key $k_{\mathrm{A}} := g^{\alpha\beta'}$ and Bob computes $k_{\mathrm{B}} := g^{\alpha'\beta}$. The attacker knows both k_{A} and k_{B}.

Attacks on the anonymous Diffie-Hellman protocol


Alice
Adversary
Bob
$\alpha \xleftarrow{R} \mathbb{Z}_q$
$u \leftarrow g^{\alpha}$
$\alpha' \xleftarrow{R} \mathbb{Z}_q$
$\beta' \xleftarrow{R} \mathbb{Z}_q$
$u' \leftarrow g^{\alpha'}$
$v \leftarrow g^{\beta}$
$\beta \xleftarrow{R} \mathbb{Z}_q$
$v' \leftarrow g^{\beta'}$
$k_a \leftarrow (v')^{\alpha} = g^{\alpha\beta'}$
$k_a, \; k_b$
$k_b \leftarrow (u')^{\beta} = g^{\alpha'\beta}$

Merkle puzzle

Can we build a secure key exchange protocol using symmetric-key primitives? The answer is yes, but the resulting protocol is very inefficient. We show how to do key exchange using a block cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{M})$. Alice and Bob want to generate a random $s \in \mathcal{M}$ that is unknown to the adversary. They use a protocol called **Merkle puzzles** (due to the same Merkle from the Merkle-Damgård hashing paradigm). The protocol, shown in Fig. 10.5, works as follows:

Protocol (Merkle puzzles)

Main tool: puzzles

- Problems that can be solved with some effort
- Example: E(k,m) a symmetric cipher with $k \in \{0,1\}^{128}$
  - **puzzle(P) = E(P, "message")** where $P = 0^{96} \| b_1 \ldots b_{32}$
  - Goal: find P by trying all 2^{32} possibilities

Protocol (Merkle puzzles)

Alice: prepare 2^{32} puzzles

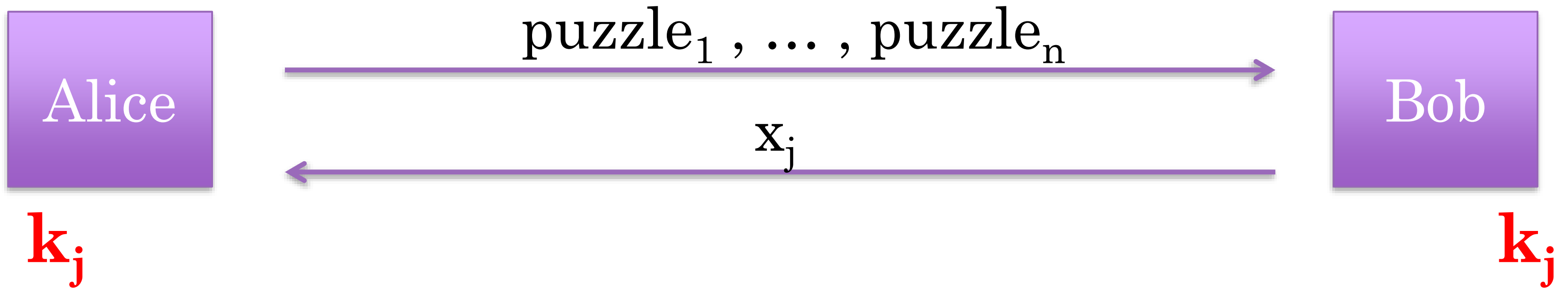
- For $i=1, \ldots, 2^{32}$ choose random $\mathbf{P_i} \in \{0,1\}^{32}$ and $\mathbf{x_i}, \mathbf{k_i} \in \{0,1\}^{128}$
  set $\text{puzzle}_i \leftarrow E(\, 0^{96} \,\|\, \mathbf{P_i} ,$ **"Puzzle # $\mathbf{x_i}$" ll $\mathbf{k_i}$** $)$
- Send $\text{puzzle}_1, \ldots, \text{puzzle}_{2^{32}}$ to Bob

Bob: choose a random puzzle_j and solve it. Obtain (x_j, k_j).

- Send x_j to Alice

Alice: lookup puzzle with number x_j. Use k_j as shared secret

Protocol (Merkle puzzles)



Alice's work: O(n) (prepare n puzzles)

Bob's work: O(n) (solve one puzzle)

Eavesdropper's work: $O(n^2)$ e.g. 2^{64} time

NYU

Protocol (Merkle puzzles)

1. Alice chooses random pairs $(k_i, s_i) \xleftarrow{R} \mathcal{K} \times \mathcal{M}$ for $i = 1, \ldots, L$. We will determine the optimal value for L later. She constructs L puzzles where puzzle P'_i is defined as a triple:

$$P'_i := \big(\ E(k_i, s_i),\ E(k_i, i),\ E(k_i, 0)\ \big).$$

Next, she sends the L puzzles in a random order to Bob. That is, she picks a random permutation $\pi \xleftarrow{R} \mathrm{Perms}[\{1, \ldots, L\}]$ and sends $(P_1, \ldots, P_L) := (P'_{\pi(1)}, \ldots, P'_{\pi(L)})$ to Bob.

Protocol (Merkle puzzles)

2. Bob picks a random puzzle $P_j = (c_1, c_2, c_3)$ where $j \overset{R}{\leftarrow} \{1, \ldots, L\}$. He solves the puzzle by brute force, by trying all keys $k \in \mathcal{K}$ until he finds one such that

$$D(k, c_3) = 0. \tag{10.4}$$

In the unlikely event that Bob finds two different keys that satisfy (10.4), he indicates to Alice that the protocol failed, and they start over. Otherwise, Bob computes $\ell \leftarrow D(k, c_2)$ and $s \leftarrow D(k, c_1)$, and sends ℓ back to Alice.

NYU

Protocol (Merkle puzzles)

3. Alice locates puzzle P'_ℓ and sets $s \leftarrow s_\ell$. Both parties now know the shared secret $s \in \mathcal{M}$.

Clearly, when the protocol terminates successfully, both parties agree on the same secret $s \in \mathcal{M}$. Moreover, when $|\mathcal{M}|$ is much larger than $|\mathcal{K}|$, the protocol is very likely to terminate successfully, because under these conditions (10.4) is likely to have a unique solution.

The work for each party in this protocol is as follows:

$$\text{Alice's work} = O(L), \qquad \text{Bob's work} = O(|\mathcal{K}|).$$

Protocol (Merkle puzzles)

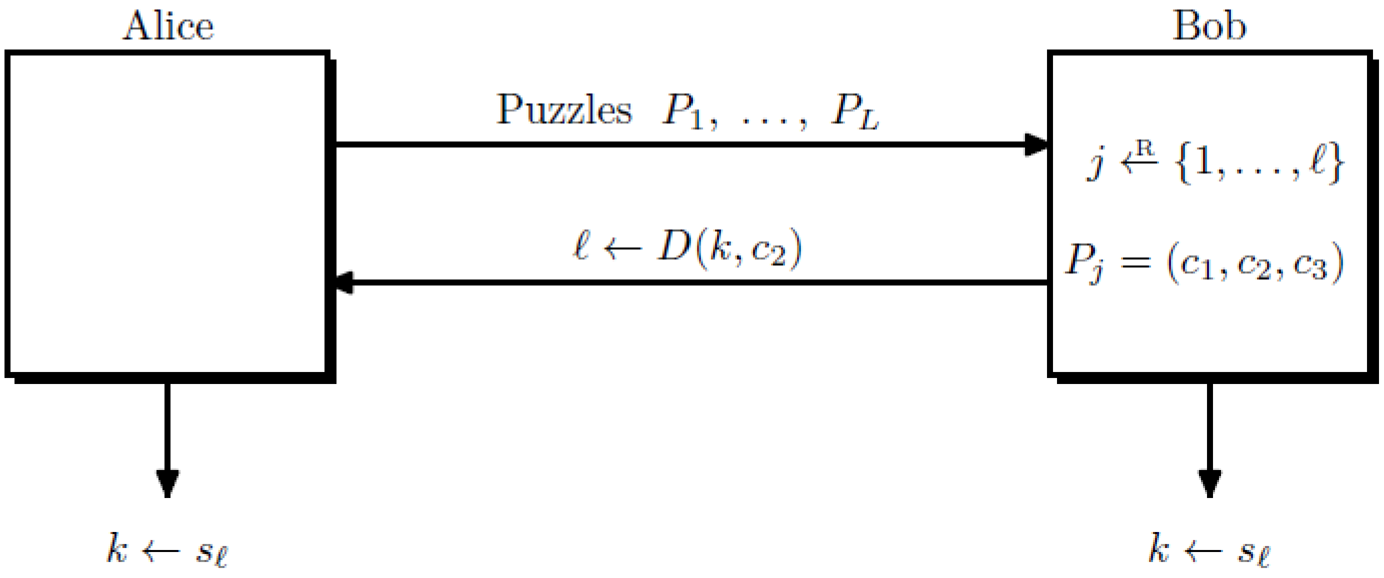

Alice
Bob
Puzzles $P_1, \ldots, P_L$
$j \xleftarrow{R} \{1, \ldots, \ell\}$
$P_j = (c_1, c_2, c_3)$
$\ell \leftarrow D(k, c_2)$
$k \leftarrow s_\ell$
$k \leftarrow s_\ell$

Figure 10.5: Merkle puzzles protocol

Security of Merkle puzzles

Hence, to make the workload for the two parties about the same we need to set $L \approx |\mathcal{K}|$. Either way, the size of L and $\mathcal{K}$ needs to be within reason so that both parties can perform the computation in a reasonable time. For example, one can set $L \approx |\mathcal{K}| \approx 2^{30}$. When using AES one can force $\mathcal{K}$ to have size 2^{30} by fixing the 98 most significant bits of the key to zero.

Security. The adversary sees the protocol transcript which includes all the puzzles and the quantity ℓ sent by Bob. Since the adversary does not know which puzzle Bob picked, intuitively, he needs to solve all puzzles until he finds puzzle P_ℓ. Thus, to recover $s \in \mathcal{M}$ the adversary must solve L puzzles each one taking $O(|\mathcal{K}|)$ time to solve. Overall, the adversary must spend time $O(L|\mathcal{K}|)$.

NYU