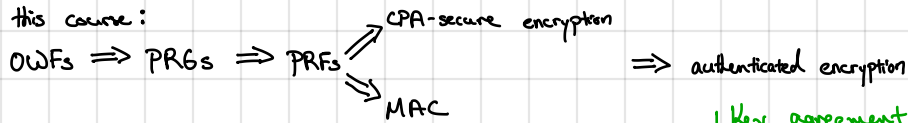


Thus far, we have assumed that parties have a shared key. Where does the shared key come from?

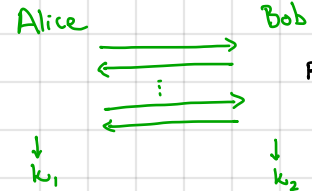
Can we do this using the tools we have developed so far?

So far in this course:



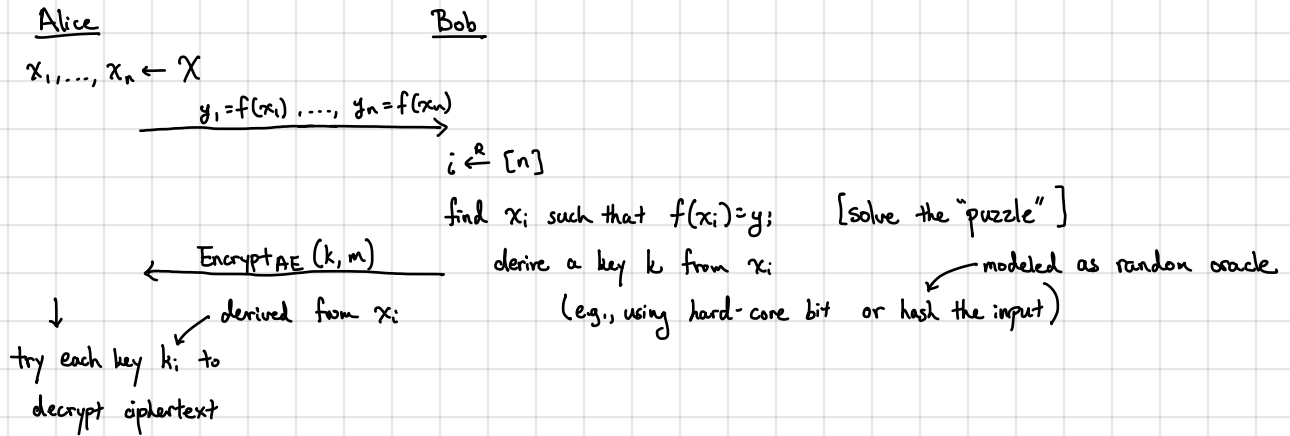
Can we show OWFs (or even OWPs)  $\Rightarrow$  key agreement?

Key agreement:



- Requirements:
- 1)  $k_1 = k_2 = k$  with high probability
  - 2) Eavesdropper cannot learn  $k_i$  (efficiently)

Merkle puzzles: Suppose  $f: X \rightarrow Y$  is an injective one-way function



Suppose it takes time  $t$  to solve a puzzle. Adversary needs time  $O(nt)$  to solve all puzzles and identify key. Honest parties work in time  $O(n+t)$ .

$\hookrightarrow$  Only provides linear gap between honest parties and adversary

Can we get a super-polynomial gap just using OWFs?  
 Can we get a super-linear gap just using OWFs?

Very difficult! [Impagliazzo-Rudich]  
 Very difficult! [Barak-Mahmoody]

$\hookrightarrow$  A positive result will require non-black-box techniques.

Impagliazzo-Rudich: Proving the existence of key-agreement that makes black-box use of OWPs implies  $P \neq NP$ .

Implication of black-box separations: Constructing secure key agreement will require more than just one-way functions  
↳ Distinction between Minicrypt and Cryptomania in "Impagliazzo's five worlds"

We will turn to algebra/number theory for new sources of hardness to build key agreement protocols.

Definition. A group consists of a set  $G$  together with an operation  $*$  that satisfies the following properties:

- Closure: If  $g_1, g_2 \in G$ , then  $g_1 * g_2 \in G$
- Associativity: For all  $g_1, g_2, g_3 \in G$ ,  $g_1 * (g_2 * g_3) = (g_1 * g_2) * g_3$
- Identity: There exists an element  $e \in G$  such that  $e * g = g = g * e$  for all  $g \in G$
- Inverse: For every element  $g \in G$ , there exists an element  $g^{-1} \in G$  such that  $g * g^{-1} = e = g^{-1} * g$

In addition, we say a group is commutative (or abelian) if the following property also holds:

- Commutative: For all  $g_1, g_2 \in G$ ,  $g_1 * g_2 = g_2 * g_1$

Notation: Typically, we will use " $\cdot$ " to denote the group operation (unless explicitly specified otherwise). We will write  $g^x$  to denote  $\underbrace{g \cdot g \cdot g \cdots g}_{x \text{ times}}$  (the usual exponential notation). We use "1" to denote the multiplicative identity ↙ called "multiplicative" notation

Examples of groups:  
 $(\mathbb{R}, +)$ : real numbers under addition  
 $(\mathbb{Z}, +)$ : integers under addition  
 $(\mathbb{Z}_p, +)$ : integers modulo  $p$  under addition [sometimes written as  $\mathbb{Z}/p\mathbb{Z}$ ]

The structure of  $\mathbb{Z}_p^*$  (an important group for cryptography):  
 $\mathbb{Z}_p^* = \{x \in \mathbb{Z}_p : \text{there exists } y \in \mathbb{Z}_p \text{ where } xy = 1 \pmod{p}\}$   
↙ here,  $p$  is prime  
↑ the set of elements with multiplicative inverses modulo  $p$

What are the elements in  $\mathbb{Z}_p^*$ ?

Bezout's identity: For all positive integers  $x, y \in \mathbb{Z}$ , there exists integers  $a, b \in \mathbb{Z}$  such that  $ax + by = \gcd(x, y)$ .

→ greatest common divisor

Corollary: For prime  $p$ ,  $\mathbb{Z}_p^* = \{1, 2, \dots, p-1\}$ .

Proof. Take any  $x \in \{1, 2, \dots, p-1\}$ . By Bezout's identity,  $\gcd(x, p) = 1$  so there exists integers  $a, b \in \mathbb{Z}$  where  $1 = ax + bp$ .  
Modulo  $p$ , this is  $ax = 1 \pmod{p}$  so  $a = x^{-1} \pmod{p}$ .

Coefficients  $a, b$  in Bezout's identity can be efficiently computed using the extended Euclidean algorithm:

Euclidean algorithm: algorithm for computing  $\gcd(a, b)$  for positive integers  $a > b$ :

relies on fact that  $\gcd(a, b) = \gcd(b, a \pmod{b})$ :

to see this: take any  $a > b$

↳ we can write  $a = b \cdot q + r$  where  $q \geq 1$  is the quotient and  $0 \leq r < b$  is the remainder

↳  $d$  divides  $a$  and  $b \iff d$  divides  $b$  and  $r$

↳  $\gcd(a, b) = \gcd(b, r) = \gcd(b, a \pmod{b})$

gives an explicit algorithm for computing  $\gcd$ : repeatedly divide:

$$\begin{array}{l} \gcd(60, 27): \quad 60 = 27(2) + 6 \quad [q=2, r=6] \rightsquigarrow \gcd(60, 27) = \gcd(27, 6) \\ \quad \quad \quad 27 = 6(4) + 3 \quad [q=4, r=3] \rightsquigarrow \gcd(27, 6) = \gcd(6, 3) \\ \quad \quad \quad 6 = 3(2) + 0 \quad [q=2, r=0] \rightsquigarrow \gcd(6, 3) = \gcd(3, 0) = 3 \end{array}$$

"rewind" to recover coefficients in Bezout's identity:

$$\begin{array}{l} \text{extended} \\ \text{Euclidean} \\ \text{algorithm} \end{array} \left\{ \begin{array}{l} 60 = 27(2) + 6 \\ 27 = 6(4) + 3 \\ 6 = 3(2) + 0 \end{array} \right. \rightarrow 3 = 27 - 6 \cdot 4 \quad \left. \begin{array}{l} 6 = 60 - 27(2) \\ 27 - (60 - 27(2))4 \\ = 27(9) + 60(-4) \end{array} \right\} \begin{array}{l} \uparrow \\ \text{coefficients} \end{array}$$

Iterations needed:  $O(\log a)$  - i.e., bit-length of the input [worst case inputs: Fibonacci numbers]

Implication: Euclidean algorithm can be used to compute modular inverses (faster algorithms also exist)

cyclic groups are commutative

defined to be the identity element

Definition. A group  $G$  is cyclic if there exists a generator  $g$  such that  $G = \{g^0, g^1, \dots, g^{|G|-1}\}$ .

Definition. For an element  $g \in G$ , we write  $\langle g \rangle = \{g^0, g^1, \dots, g^{|G|-1}\}$  to denote the set generated by  $g$  (which need not be the entire set). The cardinality of  $\langle g \rangle$  is the order of  $g$  (i.e., the size of the "subgroup" generated by  $g$ )

Example. Consider  $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$ . In this case,

$\langle 2 \rangle = \{1, 2, 4\}$  [2 is not a generator of  $\mathbb{Z}_7^*$ ]  $\text{ord}(2) = 3$

$\langle 3 \rangle = \{1, 3, 2, 6, 4, 5\}$  [3 is a generator of  $\mathbb{Z}_7^*$ ]  $\text{ord}(3) = 6$

$\hookrightarrow$  means that  $g^{\text{ord}(g)} = 1$

Lagrange's Theorem. For a group  $G$ , and any element  $g \in G$ ,  $\text{ord}(g) \mid |G|$  (the order of  $g$  is a divisor of  $|G|$ ).

$\hookrightarrow$  For  $\mathbb{Z}_p^*$ , this means that  $\text{ord}(g) \mid p-1$  for all  $g \in G$

Corollary (Fermat's Theorem): For all  $x \in \mathbb{Z}_p^*$ ,  $x^{p-1} = 1 \pmod{p}$

Proof.  $|\mathbb{Z}_p^*| = |\{1, 2, \dots, p-1\}| = p-1$

$\checkmark$  for integer  $k$

By Lagrange's Theorem,  $\text{ord}(x) \mid p-1$  so we can write  $p-1 = k \cdot \text{ord}(x)$  and so  $x^{p-1} = (x^{\text{ord}(x)})^k = 1^k = 1 \pmod{p}$

Implication: Suppose  $x \in \mathbb{Z}_p^*$  and we want to compute  $x^y \in \mathbb{Z}_p^*$  for some large integer  $y \gg p$

$\hookrightarrow$  We can compute this as

$$x^y = x^{y \pmod{p-1}} \pmod{p}$$

since  $x^{p-1} = 1 \pmod{p}$

$\hookrightarrow$  Specifically, the exponents operate modulo the order of the group

$\hookrightarrow$  Equivalently: group  $\langle g \rangle$  generated by  $g$  is isomorphic to the group  $(\mathbb{Z}_f, +)$  where  $f = \text{ord}(g)$

$$\langle g \rangle \cong (\mathbb{Z}_f, +)$$

$$g^x \mapsto x$$

Notation:  $g^x$  denotes  $\overbrace{g \cdot g \cdots g}^{x \text{ times}}$

$g^{-x}$  denotes  $(g^x)^{-1}$  [inverse of group element  $g^x$ ]

$g^{x^{-1}}$  denotes  $g^{(x^{-1})}$  where  $x^{-1}$  computed mod  $\text{ord}(g)$  — need to make sure this inverse exists!

Computing on group elements: In cryptography, the groups we typically work with will be large (e.g.,  $2^{256}$  or  $2^{1024}$ )

- Size of group element (# bits):  $\sim \log |G|$  bits (256 bits / 2048 bits)

- Group operations in  $\mathbb{Z}_p^*$ :  $\log p$  bits per group element

addition of mod  $p$  elements:  $O(\log p)$

multiplication of mod  $p$  values: naively  $O(\log^2 p)$

Karatsuba  $O(\log^{1.71} p)$

Schönhage-Strassen (GMP library):  $O(\log p \log \log p \log \log \log p)$

best algorithm  $O(\log p \log \log p)$  [2019]

$\hookrightarrow$  not yet practical ( $> 2^{4096}$  bits to be faster...)

exponentiation: using repeated squaring:  $g, g^2, g^4, g^8, \dots, g^{\log_2 p}$ , can implement using  $O(\log p)$

multiplications [ $O(\log^3 p)$  with naive multiplication]

$\hookrightarrow$  time/space trade-offs with more precomputed values

division (inversion): typically  $O(\log^2 p)$  using Euclidean algorithm (can be improved)