AMERICAN UNIVERSITY OF BEIRUT

Applied Cryptography

CMPS 297AD/396AI

Fall 2025

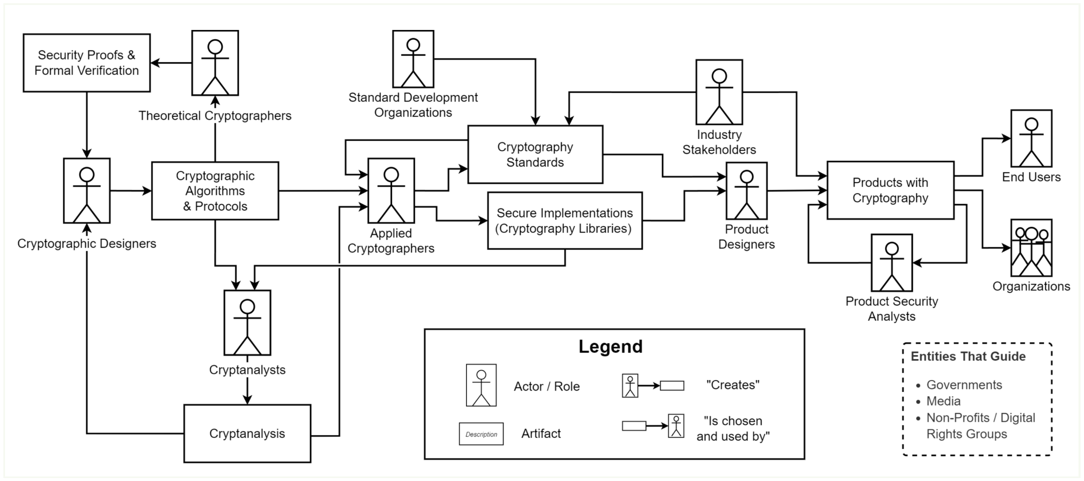
Part 1: Provable Security

1.7: Hard Problems & Diffie-Hellman

Nadim Kobeissi

https://appliedcryptography.page

How it's made



Fischer et al., The Challenges of Bringing Cryptography from Research Papers to Products: Results from an Interview Study with Experts, USENIX Security 2024

Cryptographic building blocks

Security goals

- **Confidentiality**: Data exchanged between Client and Server is only known to those parties.
- **Authentication**: If Server receives data from Client, then Client sent it to Server.
- **Integrity**: If Server modifies data owned by Client, Client can find out.

Examples

- **Confidentiality**: When you send a private message on Signal, only you and the recipient can read the content.
- **Authentication**: When you receive an email from your boss, you can verify it actually came from them.
- **Integrity**: Your computer can verify that software update downloads haven't been tampered with during transmission.

Security goals: more examples

- **TLS (HTTPS)** ensures that data exchanged between the client and the server is confidential and that parties are authenticated.
  - Allows you to log into gmail.com without your ISP learning your password.
- **FileVault 2** ensures data confidentiality and integrity on your MacBook.
  - Prevents thieves from accessing your data if your MacBook is stolen.
- **Signal** implements post-compromise security, an advanced security goal.
  - Allows a conversation to “heal” in the event of a temporary key compromise.
  - More on that later in the course.

Why bother?

- Can’t we just use access control?
- Strictly speaking, usernames and passwords can be implemented without cryptography...
- Server checks if the password matches, or if the IP address matches, etc. before granting access.
- What’s so bad about that?

The Problem with Traditional Access Control

- Requires trusting the server completely
- No protection during transmission
- No way to verify integrity
- No way to establish trust between strangers

The magic of cryptography

Cryptography lets us achieve what seems impossible

- Secure communication over insecure channels
- Verification without revealing secrets
- Proof of computation without redoing it

Section 1

Hard Problems

Hard problems

- Cryptography is largely about equating the security of a system to the difficulty of solving a math problem that is thought to be computationally very expensive.
- With cryptography, we get security systems that we can literally mathematically prove as secure (under assumptions).
- Also, this allows for actual magic.
  - Alice and Bob meet for the first time in the same room as you.
  - You are listening to everything they are saying.
  - Can they exchange a secret without you learning it?

Time for actual magic

Alice

Alice's "public key"

$g^a \bmod p$ →

Bob's "public key"

← $g^b \bmod p$

Bob

Knows public $g = 2$
Knows public prime p
Generates random a

Alice's "private key"

$$K = g^{ab} \bmod p = g^{ba} \bmod p$$

"Shared secret"

Knows public $g = 2$
Knows public prime p
Generates random b

Bob's "private key"

Why does this work?

- Listener only knows g, p, g^a, g^b
- Assuming a, b, p are large enough and p is prime...
- There is no known feasible computation to obtain K.

No known feasible computation

- The discrete logarithm problem:
  - Given a finite cyclic group G, a generator $g \in G$, and an element $h \in G$, find the integer x such that $g^x = h$
- In more concrete terms:
  - Let p be a large prime and let g be a generator of the multiplicative group $\mathbb{Z}_p^*$ (all nonzero integers modulo p).
  - Given:
    - $g \in \mathbb{Z}_p^*, h \in \mathbb{Z}_p^*$
    - Find $x \in \{0, 1, \ldots, p-2\}$ such that $g^x \equiv h \pmod{p}$
  - This problem is believed to be computationally hard when p is large and g is a primitive root modulo p.
    - "Believed to be" = we don't know of any way to do it that doesn't take forever, unless we have a strong, stable quantum computer (Shor's algorithm)

Time for more actual magic

- **Zero-knowledge proofs** allow you to prove that you know a secret without revealing any information about it.
- They built "zero-knowledge virtual machines" where you can execute an entire program that runs as a zero-knowledge proof.
- ZKP battleship game: server proves to the players that its output to their battleship guesses is correct, without revealing any additional information (e.g. ship location).

Battleship board game. Source: Hasbro

Hard problems

Asymmetric Primitives

- Diffie-Hellman, RSA, ML-KEM, etc.
- "Asymmetric" because there is a "public key" and a "private key" for each party.
- Algebraic, assume the hardness of mathematical problems (as seen just now.)

Symmetric Primitives

- AES, SHA-2, ChaCha20, HMAC...
- "Symmetric" because there is one secret key.
- Not algebraic but unstructured, but on their understood resistance to n years of cryptanalysis.
- Can act as substitutes for assumptions in security proofs!
  - Example: hash function assumed to be a "random oracle"

Hard problems

- Hard computational problems are the cornerstone of modern cryptography.
- These are problems for which even the best algorithms wouldn't find a solution before the sun burns out.
- They provide the security foundation for cryptographic schemes.
- Without hard problems, most of our encryption systems would collapse.

The rise of computational complexity theory

Computational Complexity Theory

Complexity theory provides the mathematical framework to understand what makes problems “hard”.

- In the 1970s, rigorous study of hard problems led to computational complexity theory.
- This field has had dramatic impacts beyond cryptography:
  - **Economics**: Computational complexity of finding Nash equilibria in game theory.
  - **Physics**: Simulating quantum many-body systems with exponential complexity.
  - **Biology**: Protein folding prediction and DNA sequence alignment algorithms.

Computational problems

Computational Problem

A question that can be answered by performing a computation.

- **Decision problems**: Questions with “yes” or “no” answers
  - Example: “Is 217 a prime number?”
- **Search problems**: Questions that require finding a specific value
  - Example: “How many instances of ‘*i*’s appear in ‘*incomprehensibilities*’?”

- Computational problems form the foundation of theoretical computer science.
- Different types of problems require different algorithmic approaches.
- The difficulty of solving these problems is central to cryptography.

Computational hardness

Computational Hardness

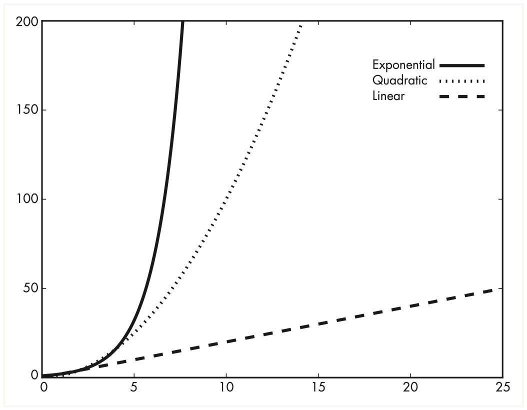
The property of computational problems for which no algorithm exists that can solve the problem in a reasonable amount of time.

- Also called **intractable problems**.
- Hardness is independent of the computing device used.
- All standard computing models are equivalent in terms of what they can compute efficiently.
- **Exception**: Quantum computers for certain problems.

- Hardness is a fundamental concept in computational complexity theory.
- Cryptography deliberately uses hard problems to create security.
- What's "hard" should remain hard regardless of hardware advances.

Measuring algorithm complexity

- To evaluate computational hardness, we need to measure an algorithm's running time.
- We typically use **asymptotic analysis** to express complexity.
- Common notation:
  - $O(n)$: Linear time.
  - $O(n^2)$: Quadratic time.
  - $O(2^n)$: Exponential time.



Complexity classes growth. Source: Serious Cryptography

Measuring algorithm complexity

- To evaluate computational hardness, we need to measure an algorithm's running time.
- We typically use **asymptotic analysis** to express complexity.
- Common notation:
  - $O(n)$: Linear time.
  - $O(n^2)$: Quadratic time.
  - $O(2^n)$: Exponential time.
- We care about how the running time grows as the input size increases.
- **Example**: An algorithm that takes n^2 operations for input size n becomes impractical as n grows large.

Categorizing computational hardness

Easy Problems

- Solvable in polynomial time.
- **Examples**: Sorting, searching.
- Running time: $O(n^c)$ for some constant c
- Generally scales reasonably with input size.
- Class P (Polynomial time).

Hard Problems

- No known polynomial-time solution.
- **Example**: Factorizing product of two large primes.
- Running time: Often exponential, e.g., $O(2^n)$
- Becomes impractical quickly as input grows.
- Includes NP-hard, NP-complete classes.

Hard problems in practice

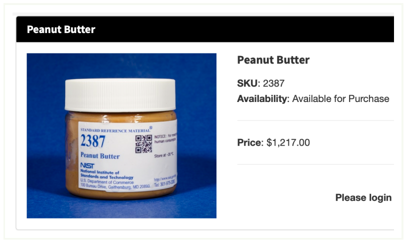
- Public-key cryptography relies on specific hard problems:
  - RSA: Integer factorization problem.
  - Diffie-Hellman: Discrete logarithm problem.
- Cryptography leverages these problems to maximize security assurance,
- The security of these schemes depends on the continued hardness of these problems.

Quantum vulnerability of hard problems

- The hard problems we rely on today (factoring, discrete logarithm) are vulnerable to quantum computers.
- Shor’s algorithm (1994) can efficiently solve both problems on a sufficiently powerful quantum computer.
- This has motivated the search for **“post-quantum”** hard problems:
  - Lattice-based cryptography (e.g., ML-KEM, formerly CRYSTALS-Kyber).
  - Hash-based cryptography.
  - Code-based cryptography.
  - Multivariate cryptography.
  - Isogeny-based cryptography.
- NIST is currently standardizing post-quantum cryptographic algorithms to replace our vulnerable systems.

What is NIST?

- **NIST** stands for the National Institute of Standards and Technology.
- It's a U.S. government agency that develops technology standards.
- In cryptography, NIST:
  - Sets security standards used worldwide.
  - Evaluates and approves cryptographic algorithms.
  - Currently leading the standardization of post-quantum cryptography.
- When NIST standardizes an algorithm, it often becomes the global industry standard.



NIST's "Standard Reference Peanut Butter", available for only $1,217 USD!

Funny things standardized by NIST

- **Standard Reference Peanut Butter**: for calibrating food testing equipment.
- **The "Odor Unit"**: for standardizing measurements of smell intensity in environmental monitoring.
- **The Standard Banana Equivalent Dose (BED)**: for comparing radiation exposure levels to the natural radiation in a banana.
- **Toilet Paper Testing**: for measuring strength, absorbency, and softness of toilet paper products.

Cryptographic algorithms standardized by NIST

- **AES (Advanced Encryption Standard)**: Selected in 2001 to replace DES, now the worldwide standard for symmetric encryption.
- **SHA-2 and SHA-3 (Secure Hash Algorithms)**: Cryptographic hash functions used for digital signatures and data integrity.
- **DSA and ECDSA**: Digital Signature Algorithms based on the discrete logarithm problem.
- **Triple DES**: An interim standard before AES that enhanced the security of the original DES.
- **ML-KEM and ML-DSA**: Recently standardized post-quantum public-key cryptography and signature schemes.

Why hard problems matter

- Hard problems **create asymmetry between legitimate users and attackers**.
- Easy in one direction, difficult in the reverse.
- Example: Easy to multiply large primes, hard to factor the product.
- This asymmetry is what enables secure communication!

What are complexity classes?

Complexity Class

A group of computational problems that share similar resource requirements (time, memory, etc.).

- **Example**: All problems solvable in $O(n^2)$ time form one class.
- Different classes represent different levels of computational difficulty.
- Understanding these classes helps us categorize cryptographic problems.

TIME complexity classes

- **TIME**$(f(n))$ = class of problems solvable in time $O(f(n))$
- Examples:
  - **TIME**(n^2) = problems solvable in $O(n^2)$ time
  - **TIME**(n^3) = problems solvable in $O(n^3)$ time
  - **TIME**(2^n) = problems solvable in $O(2^n)$ time
- **Key insight**: If you can solve a problem in $O(n^2)$ time, you can also solve it in $O(n^3)$ time.
- Therefore: **TIME**$(n^2) \subseteq$ **TIME**$(n^3) \subseteq$ **TIME**$(n^4) \subseteq \ldots$

The class P (Polynomial time)

Class P

The union of all **TIME**(n^k) classes for all constants k.

- P = **TIME**$(n)\cup$ **TIME**$(n^2)\cup$ **TIME**$(n^3) \cup \ldots$
- Contains all problems solvable in polynomial time.
- Generally considered “efficiently solvable”.

- Most practical algorithms we use daily are in class P.
- Examples: Sorting, searching, basic arithmetic.
- Cryptography often relies on problems **not** in P!

SPACE complexity classes

- Time isn't everything—memory usage matters too!
- A single memory access can be orders of magnitude slower than CPU operations.
- **SPACE**$(f(n))$ = class of problems solvable using $O(f(n))$ bits of memory
- Examples:
  - **SPACE**(n) = problems using $O(n)$ memory
  - **SPACE**(n^2) = problems using $O(n^2)$ memory
- **PSPACE** = union of all **SPACE**(n^k) for constants k

Relationship between TIME and SPACE

- **Key insight**: Any algorithm running in time $f(n)$ uses at most $f(n)$ memory.
- Why? You can write at most one bit per time unit.
- Therefore: $\textbf{TIME}(f(n)) \subseteq \textbf{SPACE}(f(n))$
- This gives us: $P \subseteq PSPACE$
- **Important**: Low memory doesn't guarantee fast execution!
  - Example: Brute-force key search uses little memory but takes forever.

The class NP (Nondeterministic Polynomial time)

Class NP

The class of decision problems for which you can **verify** a solution in polynomial time, even if finding the solution is hard.

- **Key insight**: Easy to check, hard to find!
- Given a potential solution, you can run a polynomial-time algorithm to verify if it's correct.
- You don't need to find the solution efficiently—only verify it efficiently.
- **Relationship**: $P \subseteq NP$ (if you can solve it quickly, you can certainly verify it quickly)

NP: A cryptographic example

Problem: Given plaintext P and ciphertext C, does there exist a key K such that $C = E(K, P)$?

- **Finding the solution**: Could take exponential time (brute-force key search)
- **Verifying a candidate solution**: Given a potential key K_0:
  1. Compute $E(K_0, P)$
  2. Check if $E(K_0, P) = C$
  3. Return "yes" if they match, "no" otherwise
- This verification runs in polynomial time!
- Therefore, this key recovery problem is in NP.

What's NOT in NP?

- **Known-ciphertext attacks**: You only have $E(K, P)$ values for random unknown plaintexts P.
  - How do you verify if candidate key K_0 is correct?
  - You don't know what the plaintexts should be!
  - Can't express this as a decision problem with efficient verification.
- **Proving absence of solutions**: "Does there exist NO solution to this problem?"
  - To verify "no solution exists," you might need to check all possible inputs.
  - If there are exponentially many inputs, this takes exponential time.
  - Therefore, proving non-existence is generally not in NP.

NP-complete problems

NP-Complete Problems

The hardest decision problems in the class NP.

- No known polynomial-time algorithms exist for worst-case instances.
- If any NP-complete problem can be solved efficiently, then **all** problems in NP can be solved efficiently.

- Discovered in the 1970s during the development of complexity theory.
- **Remarkable discovery**: All NP-complete problems are fundamentally equally hard!
- Examples: Boolean satisfiability (SAT), traveling salesman problem, graph coloring.

Why are NP-complete problems equally hard?

- **Key insight**: You can *reduce* any NP-complete problem to any other NP-complete problem.
- **Reduction**: Transform one problem into another in polynomial time.
  - If you can solve problem B efficiently, you can solve problem A efficiently too.
- **Mathematical equivalence**: Different NP-complete problems may look completely different but are fundamentally the same from a computational perspective.
- **Consequence**: Solving any single NP-complete problem efficiently would solve *all* problems in NP efficiently!
  - This would prove that P = NP (one of the biggest open questions in computer science).

The remarkable equivalence of NP-complete problems

These problems look completely different...

Boolean Logic

Can you set variables to make this formula true?
$(x_1 \lor \neg x_2) \land (x_2 \lor x_3) \land \ldots$

Travel Planning

What's the shortest route visiting all cities exactly once?

Sudoku Puzzles

Can you fill this 9×9 grid following the rules?

...but they're computationally identical!

Concrete examples of equivalent problems

- **3-SAT** (Boolean satisfiability): Given a logical formula, can you set the variables to make it true?
  - Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge \ldots$
- **Traveling Salesman Problem**: Given cities and distances, what's the shortest route visiting each city exactly once?
  - Looks like a geometry/optimization problem!
- **Graph Coloring**: Can you color a graph's vertices with k colors so no adjacent vertices share a color?
  - Looks like a combinatorial puzzle!
- **Subset Sum**: Given a set of integers, is there a subset that sums to exactly k?
  - Looks like an arithmetic problem!

The magic of reductions

Problem Reduction

A polynomial-time transformation that converts any instance of problem A into an equivalent instance of problem B.

- You can transform **any** Sudoku puzzle into a Boolean logic formula!
  - The Sudoku has a solution $\Leftrightarrow$ the formula is satisfiable
- You can transform **any** traveling salesman instance into a graph coloring problem!
- You can transform **any** Boolean formula into a subset sum problem!
- These transformations preserve the "yes/no" answer and run in polynomial time.
- **Mind-blowing consequence**: Solve Sudoku efficiently = solve all of theoretical computer science!

Real-world impact of this equivalence

- **Good news**: Any algorithmic breakthrough on one NP-complete problem immediately applies to thousands of others!
  - Better SAT solvers $\Rightarrow$ better protein folding, circuit design, AI planning...
- **Sobering reality**: 50+ years of computer science research suggests these problems are fundamentally hard.
  - Despite massive incentives (millions in prize money, practical applications worth billions)
- **Cryptographic relevance**: We rely on NP-complete problems being hard for certain security models.
  - Though most practical cryptography uses different hard problems (factoring, discrete log)
- **Universal truth**: The computational universe has these deep, hidden connections that unite seemingly unrelated problems.

Fun fact: Nintendo games are NP-hard!

- **Games proven NP-hard**[a]:
  - Super Mario Bros. 1–3, The Lost Levels, Super Mario World
  - Donkey Kong Country 1–3
  - All classic Legend of Zelda games
  - All classic Metroid games
  - All classic Pokémon role-playing games
- **The catch**: "Generalized versions" with arbitrarily large levels.
  - Real Nintendo levels are designed to be solvable by humans.
  - But the **mathematical structure** of these games is inherently complex.
- **Cool insight**: Video games naturally encode complex computational problems!

[a] https://appliedcryptography.page/papers/nintendo-hard.pdf

The P vs. NP Problem

The P vs. NP Problem

One of the most important unsolved problems in computer science and mathematics.

- **Question**: Does P = NP?
- **Translation**: Are there problems that are easy to verify but fundamentally hard to solve?

- If you could solve **any** NP-complete problem in polynomial time, then you could solve **all** NP problems in polynomial time.
- This would mean P = NP.
- **Intuition says**: Surely some problems are easy to check but hard to find!
- **Example**: Brute-force key recovery seems inherently exponential-time...
- **Reality**: No one has proved this mathematically!

The million-dollar question

- The **Clay Mathematics Institute** offers $1,000,000 for solving P vs. NP.
- One of seven "Millennium Prize Problems".
- Renowned complexity theorist Scott Aaronson called it *"one of the deepest questions that human beings have ever asked"*.
- **To win**: Prove either $P = NP$ or $P \neq NP$.
- Over 50 years of research, no solution yet!

What if P = NP?

The cryptographic apocalypse scenario

- If P = NP, then **any easily checked solution would also be easy to find**.
- **Symmetric cryptography** would be completely broken:
  - Key recovery becomes polynomial-time.
  - AES, ChaCha20, all symmetric ciphers become useless.
- **Hash functions** would be invertible in polynomial time:
  - Finding preimages becomes easy.
  - Digital signatures, password storage, all broken.
- **All of modern cryptography** would collapse overnight!
- **But also**: We could solve protein folding, optimize supply chains perfectly, solve climate modeling...

Why we don't panic

- **Overwhelming consensus**: Most complexity theorists believe $P \neq NP$.
- **Intuitive reasoning**: Problems that look hard actually **are** hard.
- **The structure of reality**: Easy-to-verify but hard-to-solve problems seem fundamental to the universe.
- **50+ years of evidence**: Despite massive incentives, no polynomial-time algorithms found for NP-complete problems.
- **Current belief**: P is a strict subset of NP, with NP-complete problems outside P.

The Challenge

- **Proving** $P = NP$: Need only one polynomial-time algorithm for one NP-complete problem
- **Proving** $P \neq NP$: Must prove no such algorithm can **ever** exist—much harder!

Why NP-complete problems don't work for cryptography

- **Tempting idea**: Base cryptography on NP-complete problems for provable security!
- **The dream**: Prove that breaking some cipher is NP-hard.
  - Security would be guaranteed as long as $P \neq NP$.
- **Reality is disappointing**: NP-complete problems are hard in the **worst case**, not the **average case**
  - The structure that makes them hard can make specific instances easy.
  - Cryptography needs problems that are hard for **random** instances.
- **What we actually use**: Problems that are probably **not** NP-hard.
  - Factoring, discrete logarithm, lattice problems.
  - Believed hard on average, but not proven NP-complete.

NP-complete vs. NP-hard

NP-Complete Problems

- Must be decision problems (yes/no answers)
- You can verify solutions in polynomial time
- **Examples**: 3-SAT, graph coloring, subset sum
- The “sweet spot” of hardness

NP-Hard Problems

- Can be any type of problem (optimization, etc.)
- May not have polynomial-time verification
- **Examples**: Traveling salesman optimization, halting problem
- Can be even harder than NP-complete!

Average-case vs. worst-case hardness

Worst-case hardness (NP-complete)

- Some instances of the problem are very hard.
- Other instances might be easy.
- **Example**: 3-SAT has hard instances, but also trivial ones.
- Not suitable for cryptography.

Average-case hardness (Crypto)

- Random instances are typically hard.
- Few (if any) easy instances.
- **Example**: Factoring random large integers.
- Perfect for cryptographic applications.

Hard Problems for Cryptography

We need problems where almost every instance is hard, not just the worst ones.

Section 2

Diffie-Hellman

Time for actual magic

Alice

Knows public $g = 2$
Knows public prime p
Generates random a

Alice's "private key"

Alice's "public key"

$$g^a \bmod p$$

Bob's "public key"

$$g^b \bmod p$$

Bob

Knows public $g = 2$
Knows public prime p
Generates random b

Bob's "private key"

$$K = g^{ab} \bmod p = g^{ba} \bmod p$$

"Shared secret"

Why does this work?

- Listener only knows g, p, g^a, g^b
- Assuming a, b, p are large enough and p is prime...
- There is no known feasible computation to obtain K.

The key exchange problem

- Alice and Bob want to communicate securely over the internet.
- They've never met before and share no secrets.
- How can they establish a shared secret key for encryption?
- Traditional approach: meet in person, exchange keys physically.
- **Problem**: This doesn't scale for the internet!

The Challenge

Create a shared secret between two parties who have never communicated before, even when an eavesdropper can see everything they send to each other.

The magic of Diffie-Hellman

- Whitfield Diffie and Martin Hellman solved this “impossible” problem.
- Their solution came one year **before** RSA (1977).
- Uses the **discrete logarithm problem** as its foundation.
- Allows two strangers to create a shared secret in public!

This was the birth of modern cryptography

What makes discrete logarithm hard?

- Remember: we need problems that are easy in one direction, hard in reverse.
- **Easy direction**: Given g and x, compute $g^x \mod p$
  - Example: $2^{10} \mod 17 = 1024 \mod 17 = 4$
- **Hard direction**: Given g, p, and $g^x \mod p$, find x
  - Example: Given $g = 2$, $p = 17$, and result $= 4$, find $x = 10$
- For small numbers, this is easy. For huge numbers (thousands of bits), it's computationally infeasible!

A simple example

Let's work with small numbers to see the pattern:

- Let $p = 17$ (a prime) and $g = 2$ (a generator)
- **Computing powers is easy**:
  - $2^1 \bmod 17 = 2$
  - $2^2 \bmod 17 = 4$
  - $2^3 \bmod 17 = 8$
  - $2^4 \bmod 17 = 16$
  - $2^5 \bmod 17 = 15$
- **Finding the exponent is harder**:
  - Given result 15, can you quickly find that the exponent was 5?
  - With small numbers: yes, by trying all possibilities
  - With 2048-bit numbers: practically impossible!

Mathematical groups: the foundation

What is a Mathematical Group?

A set of elements with an operation that follows specific rules.

- Think of it as a **mathematical playground** with consistent rules.
- For cryptography, we use $\mathbb{Z}_p^*$: numbers $\{1, 2, 3, \ldots, p-1\}$ with multiplication mod p.

- **Example**: $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$ with multiplication mod 5
  - $3 \times 4 = 12 \bmod 5 = 2$
  - $2 \times 3 = 6 \bmod 5 = 1$
- The "rules" ensure the math works consistently for cryptography.

Group rules (simplified)

For our cryptographic group $\mathbb{Z}_p^*$, these rules always hold:

- **Closure**: Multiplying any two elements gives another element in the group
  - In $\mathbb{Z}_5^*$: $2 \times 3 = 1$ (still in the group!)
- **Identity**: There's a special element (1) that doesn't change others
  - $1 \times 4 = 4$, $1 \times 2 = 2$, etc.
- **Inverses**: Every element has a "partner" that multiplies to 1
  - In $\mathbb{Z}_5^*$: $2 \times 3 = 1$, so 2 and 3 are inverses
- **Associativity**: $(a \times b) \times c = a \times (b \times c)$

Why care? These rules guarantee that our cryptographic operations will behave predictably!

Generators: the special elements

Generator

An element g whose powers $g^1, g^2, g^3, \ldots$ produce every element in the group.

- In $\mathbb{Z}_5^*$, let's try $g = 2$:
  - $2^1 \bmod 5 = 2$
  - $2^2 \bmod 5 = 4$
  - $2^3 \bmod 5 = 3$
  - $2^4 \bmod 5 = 1$
- We got $\{2, 4, 3, 1\}$ - that's all elements! So $g = 2$ is a generator.
- **Generators are crucial**: They let us express every group element as a power of g.

The discrete logarithm problem (DLP)

Discrete Logarithm Problem

Given g, p, and $h = g^x \mod p$, find the secret exponent x.

- **"Discrete"** because we work with integers, not real numbers
- **"Logarithm"** because we're finding the exponent (like $\log_2(8) = 3$)
- **Example**: Given $g = 2$, $p = 17$, $h = 8$, find x such that $2^x \equiv 8 \pmod{17}$
  - Answer: $x = 3$ (since $2^3 = 8$)
  - Easy with small numbers, hard with large ones!
- For cryptographic-sized numbers (2048+ bits), no efficient algorithm is known.

DLP vs. factoring: equally hard

Factoring Problem

- Given $N = p \times q$, find p and q
- Used in RSA (1977)
- Well-known, intuitive

Discrete Logarithm

- Given $g^x \mod p$, find x
- Used in Diffie-Hellman (1976)
- Less intuitive, more mathematical

- **Security equivalence**: n-bit factoring $\approx$ n-bit discrete logarithm
- Both are vulnerable to Shor's quantum algorithm
- Both are **not** known to be NP-hard
- Algorithms for both problems share similar techniques

Diffie-Hellman: the mathematical version

Setup: Alice and Bob agree on public values g (generator) and p (large prime)

1. **Alice**: Chooses secret a, computes $A = g^a \mod p$, sends A to Bob
2. **Bob**: Chooses secret b, computes $B = g^b \mod p$, sends B to Alice
3. **Alice**: Computes shared secret $S = B^a \mod p = (g^b)^a \mod p = g^{ab} \mod p$
4. **Bob**: Computes shared secret $S = A^b \mod p = (g^a)^b \mod p = g^{ab} \mod p$

Result: Alice and Bob both have $S = g^{ab} \mod p$ without ever sharing a or b!

Diffie-Hellman example with small numbers

Public parameters: $g = 2$, $p = 17$

1. **Alice**: Picks secret $a = 6$
   - Computes $A = 2^6 \bmod 17 = 64 \bmod 17 = 13$
   - Sends $A = 13$ to Bob
2. **Bob**: Picks secret $b = 10$
   - Computes $B = 2^{10} \bmod 17 = 1024 \bmod 17 = 4$
   - Sends $B = 4$ to Alice
3. **Both compute shared secret**:
   - Alice: $S = 4^6 \bmod 17 = 4096 \bmod 17 = 9$
   - Bob: $S = 13^{10} \bmod 17 = \ldots = 9$

Shared secret: $S = 9$ (which equals $2^{6\times 10} \bmod 17$)

The computational Diffie-Hellman (CDH) problem

Computational Diffie-Hellman (CDH) Problem

Given $g^a \bmod p$ and $g^b \bmod p$, compute the shared secret $g^{ab} \bmod p$ without knowing the secret values a and b.

- **Motivation**: Even if an eavesdropper captures the public values g^a and g^b, they shouldn't be able to determine the shared secret g^{ab}.
- **Example**: Given $A = 13$ and $B = 4$ from our earlier example, can you compute $S = 9$?
  - Without knowing $a = 6$ and $b = 10$, this becomes very difficult!
- **Real-world relevance**: This is exactly what an attacker faces when trying to break Diffie-Hellman.

CDH vs. DLP: the relationship

- **Key insight**: If you can solve DLP, then you can also solve CDH.
  - Given g^a and g^b, use DLP to find a and b
  - Then compute g^{ab} directly
- **Mathematical relationship**: DLP is **at least as hard** as CDH.
  - CDH $\leq$ DLP (CDH reduces to DLP)
- **Open question**: Is CDH at least as hard as DLP?
  - We don't know if solving CDH allows you to solve DLP!
  - Maybe there's a clever way to compute g^{ab} without finding a and b
- **Security assumption**: We assume CDH is hard even if it's easier than DLP.

The decisional Diffie-Hellman (DDH) problem

Decisional Diffie-Hellman (DDH) Problem

Given $g^a \mod p$, $g^b \mod p$, and a value X that is either:

- $g^{ab} \mod p$ (the real shared secret), or
- $g^c \mod p$ for some random c

...determine which one X is (each choice has probability 1/2).

- **Why do we need this?** Indistinguishability!
  - What if an attacker can compute the first 32 bits of g^{ab}?
  - CDH isn't completely broken, but the attacker learned something.
  - This partial information might compromise application security.
- **DDH ensures**: The shared secret g^{ab} is **indistinguishable** from a random group element.

DDH vs. CDH: the hierarchy

- **Key relationship**: If you can solve CDH, then you can solve DDH.
  - Given (g^a, g^b, X), use CDH to compute g^{ab}
  - Check if $X = g^{ab}$; if yes, then X is the real shared secret
- **Hardness hierarchy**: DDH $\leq$ CDH $\leq$ DLP
  - DDH is fundamentally **easier** than CDH.
  - CDH is (probably) easier than DLP.
- **Surprising fact**: DDH is **not hard** in certain groups!
  - In $\mathbb{Z}_p^*$, DDH can be broken using pairing-based techniques.
  - But CDH remains hard in the same group.
- **Solution**: Use elliptic curve groups where DDH is believed hard.

Why DDH matters in cryptography

- **Indistinguishability**: DDH ensures that shared secrets “look random”.
  - Critical for encryption schemes and key derivation.
  - Prevents attackers from learning partial information.
- **Security proofs**: Many cryptographic protocols prove security under DDH.
  - ElGamal encryption.
  - Cramer-Shoup cryptosystem.
  - Various authenticated key exchange protocols.
- **Real-world impact**: Even though DDH is “weaker” than CDH, it’s one of the most studied and used assumptions.
  - Provides stronger security guarantees for applications.
  - Enables more sophisticated cryptographic constructions.

Real-world Diffie-Hellman

- **TLS/HTTPS**: Your browser uses Diffie-Hellman to establish secure connections.
- **Signal**: Uses elliptic-curve Diffie-Hellman for key exchange.
- **SSH**: Secure shell connections use Diffie-Hellman for key agreement.
- **VPNs**: Many VPN protocols rely on Diffie-Hellman for establishing tunnels.

Modern Diffie-Hellman Variants

- **Elliptic Curve Diffie-Hellman (ECDH)**: Same idea, different mathematical group.
- **Post-quantum alternatives**: New key exchange methods for the quantum era.

More on both of the above in future course topics!

Diffie-Hellman key exchange in practice

How does this actually work in the real world?

1. **Parameter generation**: Choose secure values for p and g
   - p must be a large prime (2048+ bits)
   - g must be a generator of a large subgroup
2. **Key generation**: Each party picks a random secret
   - Alice picks a randomly from $\{1, 2, \ldots, p-2\}$
   - Bob picks b randomly from $\{1, 2, \ldots, p-2\}$
3. **Public key computation**: Each party computes their public value
4. **Key exchange**: Public values are sent over the network
5. **Shared secret derivation**: Each party computes the final shared secret

TLS handshake: Diffie-Hellman in action

When you visit https://gmail.com, here's what happens:

1. **Client Hello**: Your browser says "I want to talk securely"
2. **Server Hello**: Gmail's server responds with its certificate and DH parameters
   - Includes p, g, and server's public value $g^b \bmod p$
3. **Client Key Exchange**: Your browser generates its own secret a and sends $g^a \bmod p$
4. **Secret computation**: Both sides compute $g^{ab} \bmod p$
5. **Key derivation**: The shared secret is used to derive encryption keys
6. **Secure communication**: All further messages are encrypted with these keys

Result: Your password is encrypted before leaving your computer!

Signal's double ratchet: DH everywhere

- **Initial key exchange**: Uses X3DH (Extended Triple DH)
  - Combines **three** DH key exchanges for security.
  - Works even when recipient is offline ("*asynchronous*" protocol).[a]
- **Ongoing communication**: Uses Double Ratchet
  - New DH key exchange for every message!
  - Provides "forward secrecy" and "post-compromise security".
  - If your phone gets compromised today, yesterday's messages remain secure.
  - If your phone recovers from compromise, tomorrow's messages are secure again.

Signal uses DH key exchange dozens, hundreds of times per conversation.

[a] Everything on this slide will be covered in much more detail later in the course.

The dark side: unauthenticated Diffie-Hellman

But there's a serious problem...

- **The vulnerability**: Basic DH has no authentication
  - Alice can't verify she's talking to Bob
  - Bob can't verify he's talking to Alice
- **The attack**: Man-in-the-middle (MITM)
  - Mallory sits between Alice and Bob
  - Alice does DH with Mallory, thinking it's Bob
  - Bob does DH with Mallory, thinking it's Alice
  - Mallory can read and modify everything!
- **Real-world impact**: This attack is practical and devastating!

Man-in-the-middle attack on DH

How Mallory breaks "secure" communication:

1. **Alice** → **Mallory**: Alice sends g^a (thinking it goes to Bob)
2. **Mallory** → **Bob**: Mallory sends g^m (Bob thinks it's from Alice)
3. **Bob** → **Mallory**: Bob sends g^b (thinking it goes to Alice)
4. **Mallory** → **Alice**: Mallory sends g^m (Alice thinks it's from Bob)
5. **Result**:
   - Alice and Mallory share secret g^{am}
   - Bob and Mallory share secret g^{bm}
   - Alice and Bob don't share any secret!
6. **Communication**: Alice encrypts with g^{am}, Mallory decrypts, reads/modifies, re-encrypts with g^{bm} for Bob

Alice and Bob never know they've been compromised!

Why MITM attacks succeed

- **Public values look random**: g^a and g^m are indistinguishable.
  - Both appear to be random group elements.
  - No way to tell if they come from the intended party.
- **No identity verification**: DH only establishes a shared secret.
  - Doesn't prove who you're sharing it with!
  - Like agreeing on a secret handshake with someone wearing a mask.
- **Active vs. passive attacks**:
  - DH protects against **passive** eavesdropping.
  - Does nothing against **active** manipulation.
- **Historical impact**: This attack has compromised real systems for decades.

Solution: Authenticated Key Exchange

Authenticated Key Exchange (AKE)

Key exchange that verifies the identity of the parties involved, preventing man-in-the-middle attacks.

- **Core idea**: Combine DH with authentication mechanisms
- **Common approaches**:
  - **Digital signatures**: Sign the DH public values (TLS).
  - **Pre-shared keys**: Use existing shared secrets (IPsec).
  - **Certificates**: Use a trusted third party (Certificate Authority in HTTPS).
  - **Password-based**: Derive authentication from passwords (SRP protocols).
- **Goal**: Ensure that Alice and Bob can verify they're really talking to each other.

TLS: authenticated DH with certificates

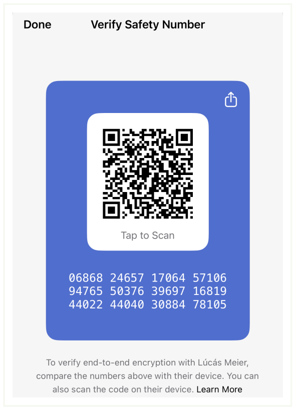
How HTTPS prevents MITM attacks:

1. **Server authentication**: Gmail sends its certificate along with g^b
   - Certificate proves “this DH value really came from gmail.com”
   - Signed by a trusted Certificate Authority (CA)
2. **Certificate verification**: Your browser checks:
   - Is the signature valid?
   - Is the CA trusted?
   - Does the certificate match “gmail.com”?
   - Has the certificate expired?
3. **If verification passes**: You know you’re really talking to Gmail
4. **If verification fails**: Browser shows scary warnings!

Result: MITM attacks become much harder (but not impossible!)

Signal: authenticated DH with fingerprints

- **The bootstrapping problem**: How do Alice and Bob initially authenticate?
  - No pre-existing certificates.
  - No trusted third parties.
- **Signal's solution**: Security numbers (fingerprints)
  - Each conversation gets a unique 60-digit number.
  - Derived from both parties' long-term identity keys.
- **Manual verification**: Users compare numbers out-of-band.
  - Read over the phone...
  - Show in person...
  - Send via different app...



Signal security number verification screen.

SSH: authenticated DH with host keys

How SSH prevents server impersonation:

- **First connection**: Server presents its "host key" along with DH public value
  - SSH shows you a fingerprint: `SHA256:ABC123...`
  - You're supposed to verify this out-of-band (but nobody does!)
- **Trust on first use (TOFU)**: Client remembers the host key
  - Stored in `~/.ssh/known_hosts`
- **Subsequent connections**: Client checks if host key matches
  - If different, gives you a heart attack: `WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!`
  - If same: Connection proceeds normally
- **User authentication**: Usually with passwords or public keys

Weakness: TOFU is vulnerable on the very first connection!

Modern implementations: elliptic curves

Traditional DH

- Uses $\mathbb{Z}_p^*$ (integers mod p)
- Requires 2048+ bit numbers
- Slower computations
- Larger public keys

Elliptic Curve DH (ECDH)

- Uses elliptic curve groups
- 256-bit keys ≈ 2048-bit traditional DH
- Much faster computations
- Smaller public keys, less bandwidth

- **Popular curves**: P-256, P-384, X25519, X448
- **Same security**: Based on elliptic curve discrete logarithm problem
- **Real-world adoption**: ECDH is now standard in TLS, Signal, etc.
- **Performance matters**: Especially important for mobile devices and IoT

The quantum threat to Diffie-Hellman

All DH variants are doomed...

- **Shor's algorithm** (1994) can break DH on quantum computers.
  - Solves discrete logarithm in polynomial time.
  - Works for both traditional DH and ECDH.
- **Timeline concerns**:
  - Large quantum computers don't exist yet.
  - But adversaries might store encrypted data now, decrypt later.
  - "Harvest now, decrypt later" attacks.
- **Post-quantum key exchange**: New algorithms under development.
  - ML-KEM (based on lattice problems)
  - SIDH/SIKE (based on isogenies, but recently broken!)
  - Code-based and hash-based alternatives

Lessons from 50 years of Diffie-Hellman

- **Elegant mathematics**: Simple idea with profound implications.
  - Two numbers raised to secret powers in a mathematical group.
- **Security requires more than math**: Authentication is crucial.
  - Pure DH is vulnerable to active attacks.
  - Real systems need identity verification.
- **Efficiency drives adoption**: Elliptic curves made DH practical everywhere.
  - Performance improvements enable new applications.
- **Future challenges**: Quantum computers will force reinvention.
  - But the core insight—shared secrets from public exchanges—will survive.
- **Cryptography is a living field**: Continuous evolution and adaptation.

From hard problems to real-world security

The journey we've traced

1. **Mathematical insight**: Discrete logarithm is hard to compute.
2. **Cryptographic innovation**: Diffie-Hellman key exchange leverages this hardness.
3. **Real-world impact**: Secure communication for billions of people daily.

This is the power of applied cryptography: transforming abstract mathematical problems into tools that help people and protect our digital lives.

AMERICAN UNIVERSITY OF BEIRUT

Applied Cryptography

CMPS 297AD/396AI

Fall 2025

Part 1: Provable Security

1.7: Hard Problems & Diffie-Hellman

Nadim Kobeissi
https://appliedcryptography.page