CS409 : Chalk and Talk

Davies-Meyer Construction for Compression Functions and Time-Space Tradeoff Attacks on Hash Functions

Kshitij Vaidya - 22b1829
Ishan Pandit - 22b2141

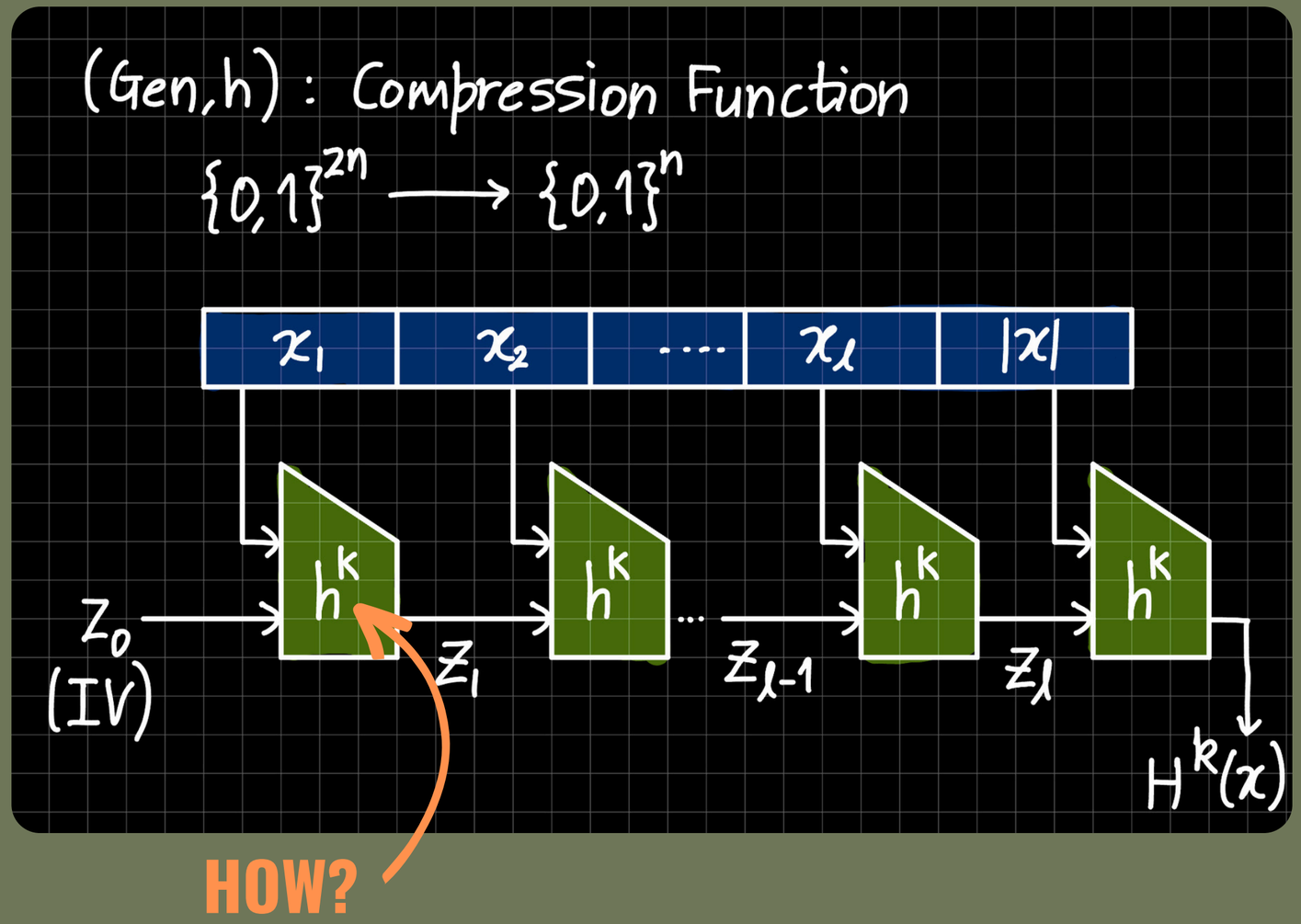
Davies-Meyer Construction for Compression Functions

Why Compression Functions?

- Take large chunks of data and break them down into smaller fixed-size chunks, allowing for storage efficiency and faster processing.
- They are a key component in the construction Hash Functions to handle arbitrary length inputs by considering fixed length “bytes” of data.
- Well-designed compression functions are used to add layers of security making it difficult to find collisions and generate specific hash outputs.

The Merkle-Damgård Transform

- This is a scheme used to build **CRHFs** from collision-resistant Compression Functions
- A message of arbitrary length is broken into fixed length pieces and fed into a compression function along with the previous “chaining value”



The Davies-Meyer Construction

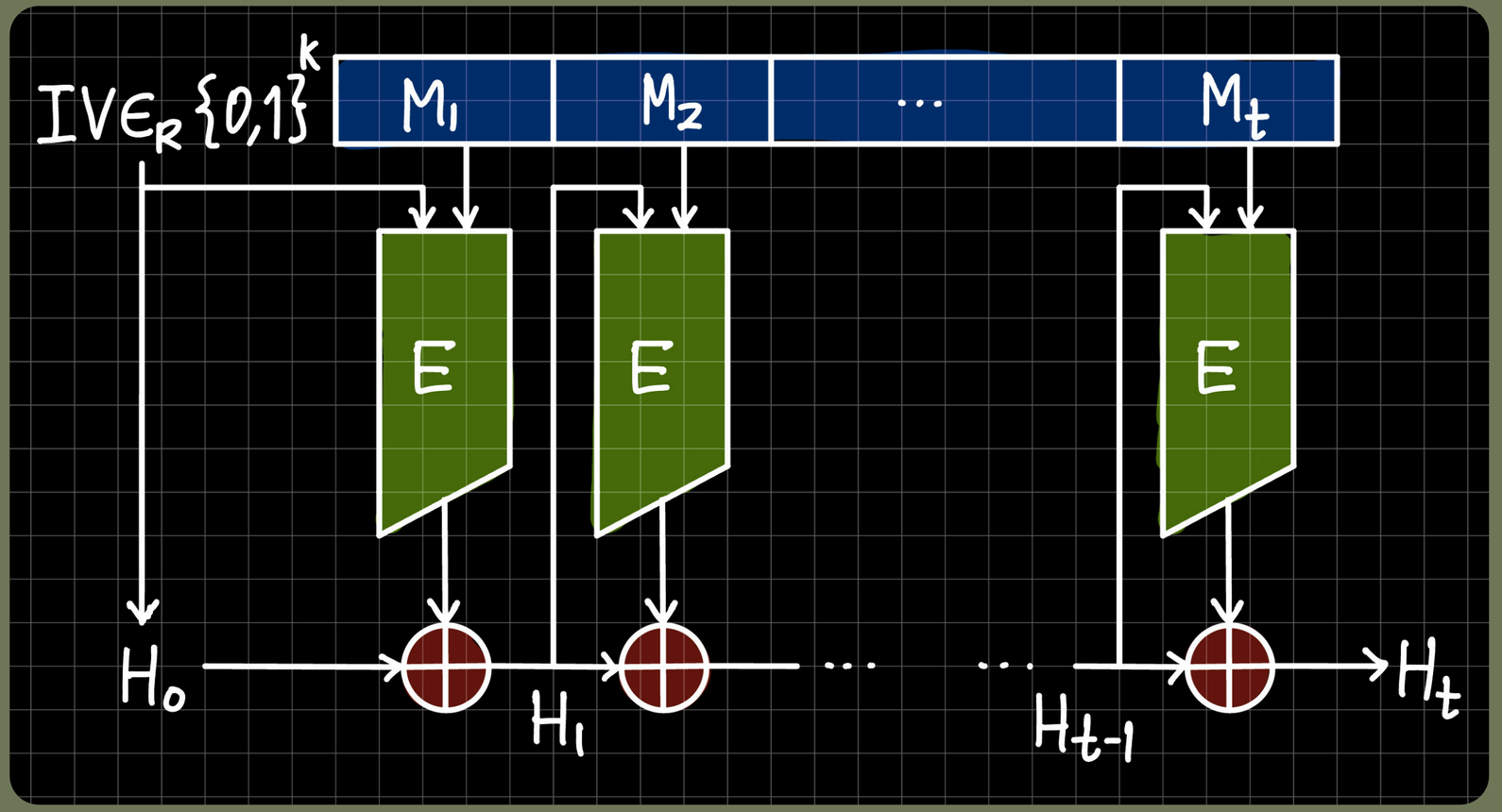
The Davies-Meyer Construction for Compression Functions provides a mechanism to build a secure one-way hash function by leveraging the properties of block ciphers. The building block of the construction relies on the following :

- A Secure Block Cipher
- The XOR Operation

This relatively straightforward algorithm is used to generate compression and hash functions that showcase a high degree of security.
It has been previously used in MD5, SHA-1 and the SHA-2 family highlighting its robustness and practical relevance.

Building the Compression Function

$$H_i = E\left(H_{i-1}, M_i\right) \oplus H_{i-1}$$



Security of the Construction...

- The security of the Davies-Meyer Construction relies on the security assumptions made for the Block Cipher, which essentially acts as a keyed PRP.
- We can treat the Block Cipher as a "Black Box" or a Random Oracle which can be queried for both Encryptions and Decryptions.

Claim 1: For a block cipher of message length m, a Random Oracle Model solving :

Given m-bit value H, find any pair (X,K) s.t $E_K(X) \oplus X = H$

has **expected runtime** of **atleast** 2^m

Claim 2: Given hash value Y, finding any message M and initial value I such that

$H(I,M) = Y$

requires 2^{m-1} calls under ROM

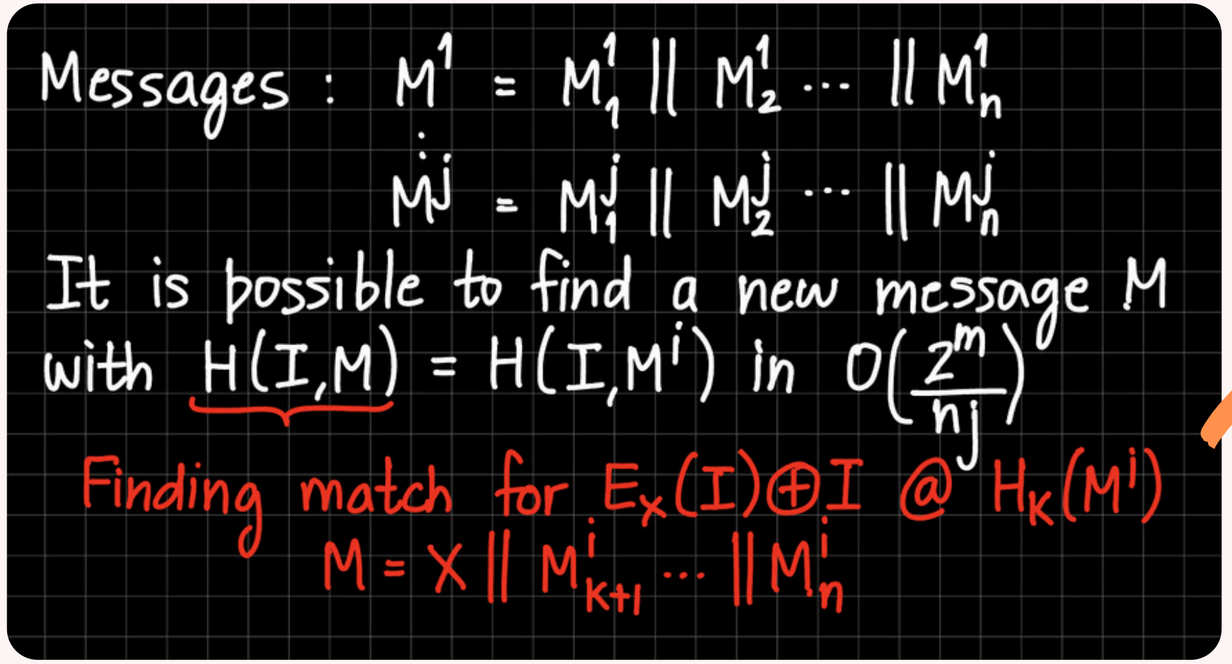
PreImage Resistance ~ Infeasibility of inverting the PRP

Collision Resistance ~ Psuedorandomness of PRP and Uniqueness by XORing with X

Helps Prove PreImage Resistance for DM

Can We Do Better?

- The current scheme becomes less secure in the situation where multiple key signatures and their messages are available. This poses a practical threat to the security of the system



Can we Prevent this Speedup?

Modified Davies-Meyer

We must make the problems faced by the forger independent. We can do this by adding a running count! The system will end up requiring more encryptions because of this but it allows us to achieve the required degree of security.

Construction 2 : Let E has k-bit Key , m-bit Message , assume $\leq 2^t$ total blocks are signed

$M := M_1 \| M_2 \cdots M_n$ of length k-t

$$H_0 = I$$

$$H_{i+1} = H_i \oplus E_{T+i \| M_{i+1}}(H_i) \quad i \leq n$$

$$H(I, M) = (T+n, H_n)$$

Practical Applications

- **MD5 (Message Digest Algorithm 5) :** used a construction similar to the Davies-Meyer
- **SHA-2 (Secure Hash Algorithm 2) :** Also built around compression functions similar of DM with adjustments to enhance security
- **Whirlpool :** Explicitly designed using the Davies-Meyer scheme and used the Rijndael (AES) as the Block Cipher

Time-Space Tradeoff Attacks on Hash Functions

What are Hash Functions?

- A cryptographic tool that converts data input into a fixed-length, unique "hash" output.
- Designed to be fast, deterministic, and resistant to reverse-engineering, ensuring that finding two inputs with the same hash is nearly impossible.
- Used to secure data, authenticate messages, digitalize signatures and protect stored passwords.

Properties of Hash Functions

For a hash function to be considered cryptographically secure:

- Deterministic: The same input will always produce the same hash output.
- Preimage Resistance: For H, infeasible to find any message M such that H(M) = H.
- Second Preimage Resistance: Given a message M1 with hash H(M1), it should be infeasible to find another message M2 such that H(M1) = H(M2).
- Collision Resistance: It should be infeasible to have different messages M1 and M2 that hash to the same value H(M1) = H(M2).
- Fast Computation: Hashing the input should be computationally efficient and fast.

SHA-256

1

Secure Hash Algorithm (256-bit)

SHA-256 is a hash algorithm which takes an input of any size and generates a 256-bit (32-byte) fixed-length output, typically represented as a 64-character hexadecimal number.

Steps of the SHA-256 algorithm:

- Padding to create 512-bit chunks.
- Initialization of output with the hash constants.
- Iteratively processing through the chunks, using a series of mathematical operations involving bitwise operations, modular additions, and logical functions.
- The final output is the 256-bit hash value.

Hash Function Attacks

- Hash functions, while designed to be secure, are susceptible to a range of attacks aimed at exploiting weaknesses in their properties.
- The primary goal of attackers is to find a way to reverse the hash (retrieve the original input) or create identical hashes from different inputs to undermine the function’s integrity.
- There are various types of attacks: Brute-Force Attacks, Birthday Attacks, Time-Space Tradeoff Attacks etc.

Time-Space Tradeoff

- The time-space tradeoff is a concept where computation time is conserved at the expense of memory or storage (or vice versa).
- In cryptography, this principle allows storage of precomputed information to reduce the computational time needed in a task (e.g. attacks).
- The tradeoff is typically between the space needed to store large tables of precomputed values (e.g., hashes) and the time saved by avoiding on-the-fly calculations during an attack i.e. Storing vs Calculating.

Principle of Tradeoff Attacks

- Tradeoff attacks are based on balancing memory storage with computational time to efficiently find hash collisions or preimages.
- By precomputing and storing values (often as tables of hashes), attackers can reduce the time required to find a hash match.
- Main types: Rainbow Table attacks and Distinguished Point attacks.
- These attacks are effective mainly against unsalted hash functions, which produce identical hashes for the same input across different instances.

Rainbow Table Attack

- Uses precomputed tables of hash chains to efficiently find hash collisions or reverse hashes back to their original input values.
- Use less computer processing time and more storage than a brute-force attack.

- **Hash Chain Generation:** Store only the starting and ending points of chains generated by repeatedly applying a hash function and a reduction function.
- **Reduction Function**: The reduction function maps hashes back to potential inputs (eg. potential passwords), allowing for the creation of a sequence of hashes.
- **Searching with Rainbow Tables**: When an attacker targets a specific hash, they check it against the endpoints in the table and simulate a chain starting from that hash to see if it matches any stored endpoints.

Distinguished Point Attack

- DP attacks store only special, identifiable points within each chain as markers.
- These points, called "**distinguished points**", are chosen based on a characteristic that makes them easy to recognize, such as a specific number of leading zeros in their binary representation.
- Thus, the computed Hash Chains are variable in length, stopping only when a hash reaches a distinguished point. They use similar reduction functions as RT Attacks.
- Each round of hashing and reducing produces a new hash, which is checked for the distinguished point criterion (matching with a DP in the table).

Mitigating Tradeoff Attacks

- **Salting Hashes:** Adding unique salts to each input ensures even identical passwords create distinct hashes, preventing precomputed attacks like Rainbow Tables by making tables for every salt combination impractical.
- **Using HMACs:** HMACs use a secret key with the hash, creating key-dependent hashes that attackers cannot replicate without the key, adding strong protection against brute-force and precomputed attacks.

Thank You!