Mini-Project: Generating Rogue Certificate via Finding Hash Collision

Gihyuk Ko

Carnegie Mellon University

Rogue Certificate

- Sotoriv et al., “MD5 Considered Harmful Today: Creating a rogue CA certificate”
- Slides available from here

SHA-256

- Secure Hash Algorithm (SHA)
  - Developed by NIST among with NSA
  - Multiple versions: SHA-0 (1993), SHA-1 (1995), SHA-2 (2001), SHA-3 (2012)

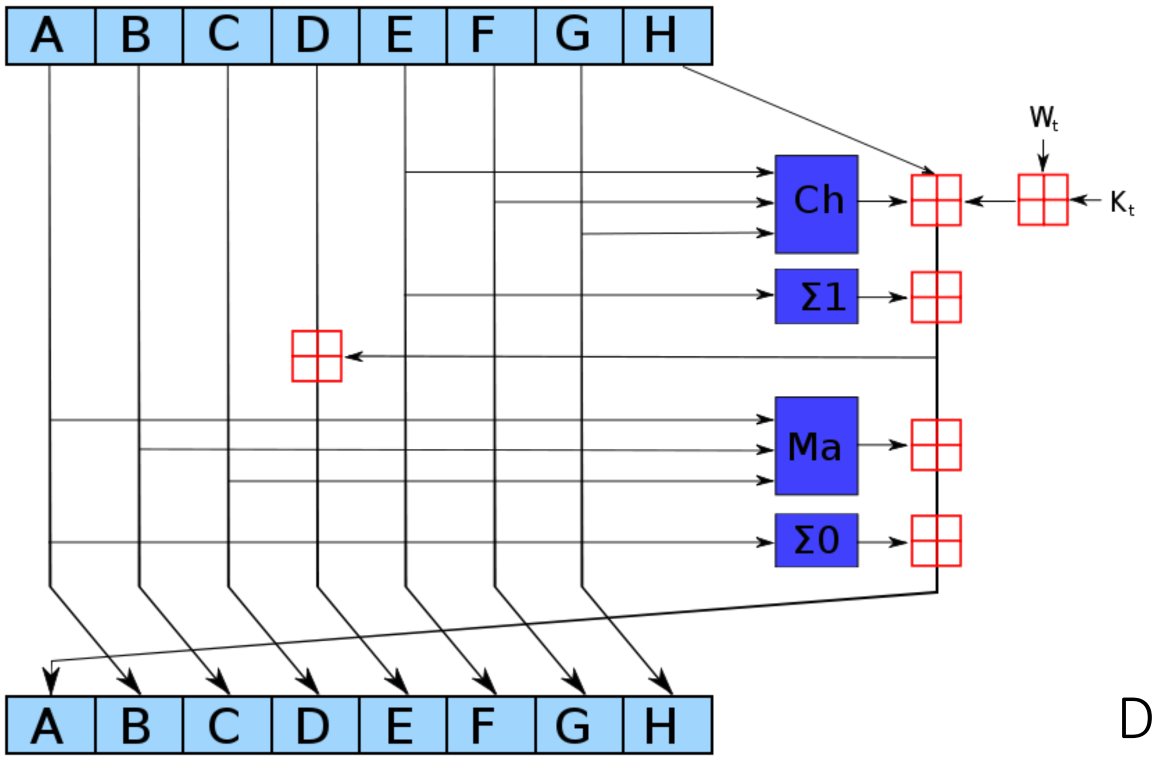
- SHA-2 family of hash functions
  - Consists of 6 hash functions with digests of 224, 256, 384, 512 bits
    - SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256

SHA-256

- In functional form, SHA-256: $\{0,1\}^* \rightarrow \{0,1\}^{256}$
- Stepwise computation of the hash
    1. Arbitrary length input is padded up to multiple of 512-bit block with special padding scheme
    2. Some preprocessing is done to the padded-up values
    3. For each block (or chunk), apply compression function
    4. Add up all results from each block(chunk)

SHA-256: Compression

- Computation done each round looks like follows:



A~H: 32-bit words

⊞ : addition mod 2^{32}

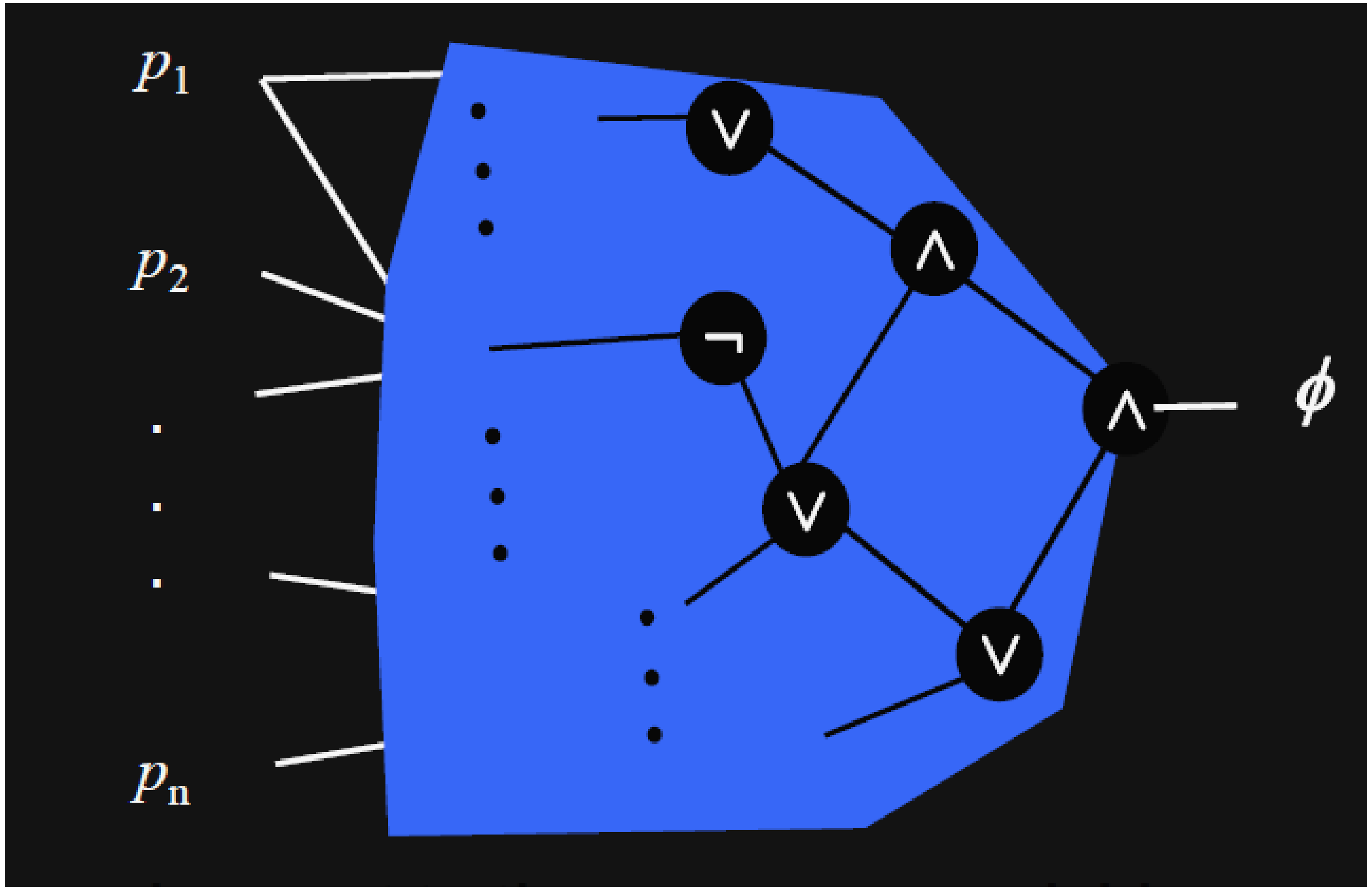
$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$
$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$
$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$
$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

Detailed version of pseudocode here

SHA-256-18*

- There is no such thing in reality! We are making up our own hash for this assignment ☺
- Two simplifications
  - 0-pad inputs to the multiple of 64 bytes
  - Reduce the number of rounds into 18 rather than 64
- `sha256_template.py` file has full specifications for SHA-256-18!

Boolean Satisfiability Problem (SAT)



Q. Is there an assignment to the variables p_1, p_2, …, p_n such that ϕ evaluates to 1?

Boolean Satisfiability Problem (SAT)

- Examples
  - $\phi_1(a, b, c) = (a \wedge \bar{b}) \vee c$

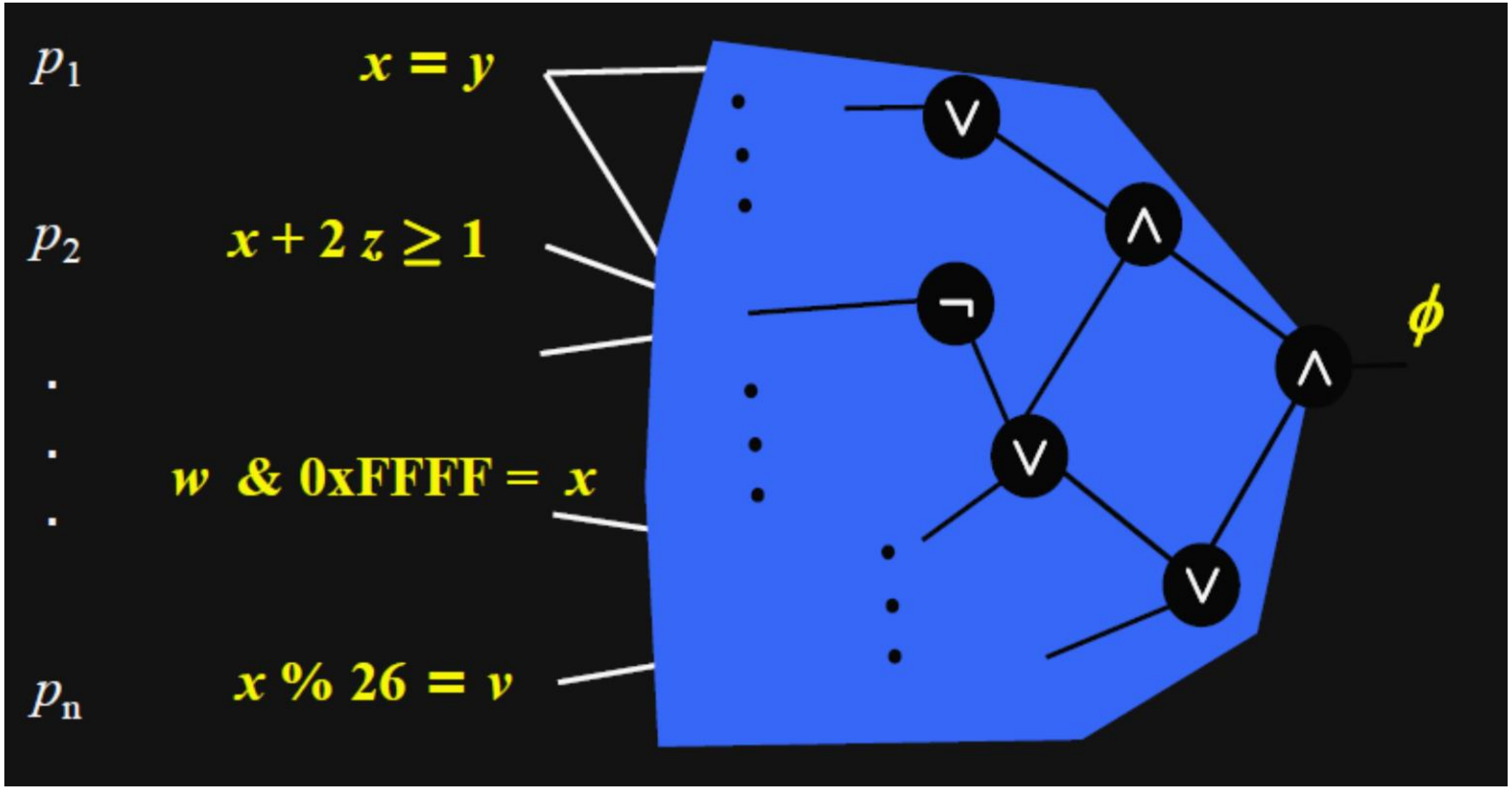
    ➔ (a, b, c): $(T, F, F), (F, T, F), \dots$
  - $\phi_2(a, b, c) = (a \vee b) \wedge \bar{c}$

    ➔ (a, b, c): $(T, T, F), (F, T, F), \dots$
  - $\phi_3(a, b, c) = a \vee b \vee c$

    ➔ (a, b, c): $(F, T, F), (F, T, F), \dots$
  - $\phi_4(a, b, c) = \phi_1 \wedge \phi_2 \wedge \phi_3$

    ➔ (a, b, c): $(T, F, F), (F, T, F), \dots$

Satisfiability Modulo Theory (SMT)



Q. Is there an assignment to the variables *x,y,z,w* s.t. ϕ evaluates to 1?

Satisfiability Modulo Theory (SMT)

- Examples
  - $\phi_1(x) = (x > 5) \land (x < 10)$

    ➔ x: 6,7,8,9
  - $\phi_2(x, y) = (x > y) \land (y > 7)$

    ➔ (x, y): (9,8), (10,9)…
  - $\phi_3(x, y, z) = (z > x) \land (z < 2y)$

    ➔ (x, y, z): (4,3,5), (6,5,9), (9,8,10)…
  - $\phi_4(a, b, c) = \phi_1 \land \phi_2 \land \phi_3$

    ➔ (x, y, z): (9,8,10), …

SMT Solvers

- Given a Boolean SMT form efficiently searches whether a solution exists or not
- Not every SMT problems are solvable efficiently!
  - 3-SAT problem is known to be NP-complete!
- Applications
  - Theorem Prover: Z3, Boogie, Dafny, …
  - Symbolic Execution: Z3, STP, Boolector, …

Z3 Theorem Prover

- A theorem prover developed by Microsoft Research
  - Available for download and install here
  - Also available through python package managers such as anaconda

- In this project, we will use Z3 with `python`
  - Other languages are okay to use, but please note that the template codes are in `python`

Demo

Goal of the Mini Project

- Using Z3, generate collision in SHA-256-18
- Using a given valid certificate, generate a valid rogue CA certificate via hash collision

Certificate Specification*

- Does not follow X.509 standard, but still has similar information inside
- Fields
  - `version, serial`
  - `sig_algorithm`
  - `issuer`
  - `validity_start, validity_end`
  - `subject_name, subject_public_key_algorithm, subject_public_key`
  - `is_ca`
  - `issuer_unique_id, subject_unique_id`
  - `signer_private_key_str`
- Full specification in `certificates_template.py` file

Requirements

1. **[8 points]** Find a collision for SHA-256-18, that one message is not a zero-padded version of the other message.
    - Find two strings s_1, s_2 such that SHA-256-18(s_1) = SHA-256-18(s_2)
        - s_1 cannot be a zero-padded version of s_2, and vice versa
    - Use Z3 to find such collision
    - You may need to find proper constraints to input to the Z3 solver
    - Be careful on what operations you may want to use in each step operation

Requirements

2. **[2 points]** Create a rogue CA certificate using the SHA-256-18 collision.
    - Given things
        - a public-private RSA key pair (class_public, class_private)
        - a valid certificate signed by Gihyuk's private key
        - Gihyuk's public key: can be used to verify
    - Certificate should have power to endorse other certificates (should be a CA certificate)
    - Detailed instructions on write-up

Final Points

- Mini project will be out around 8pm EST today
- Please submit your code in one file named:

```
[project]_[andrewid].py
```

  - Please include your text submission inside the code in a comment form!
  - If you want to make a separate .pdf for text submission, that’s also okay
- No grace days are available to use for the mini project, so start early!
- It is NOT permitted in any case to show or share source code: please work out individually!

Thanks!