Cryptanalysis of the GOST Hash Function

Florian Mendel, Norbert Pramstaller,
and Christian Rechberger

Institute for Applied Information Processing and Communications (IAIK) - Krypto Group

Faculty of Computer Science
Graz University of Technology

Outline

- Motivation
- Description of GOST
- A preimage attack on the GOST hash function (FSE 2008)
  - A pseudo-preimage attack on the compression function
  - A preimage attack on the hash function
- Improving the attack (work in progress)
  - A fixed-point in the GOST block cipher
  - Improving the preimage attack on the hash function
  - A collision attack on the hash function
- Conclusion and Future Work

Motivation

- Russian government standard (GOST-R-34.11-94)
- Russian Digital Signature Algorithm
  (GOST-R-34.10-94 and GOST R 34.10-2001)
- Specified in several RFCs
- Implemented in SSL (openSSL)
- ….

Security requirements

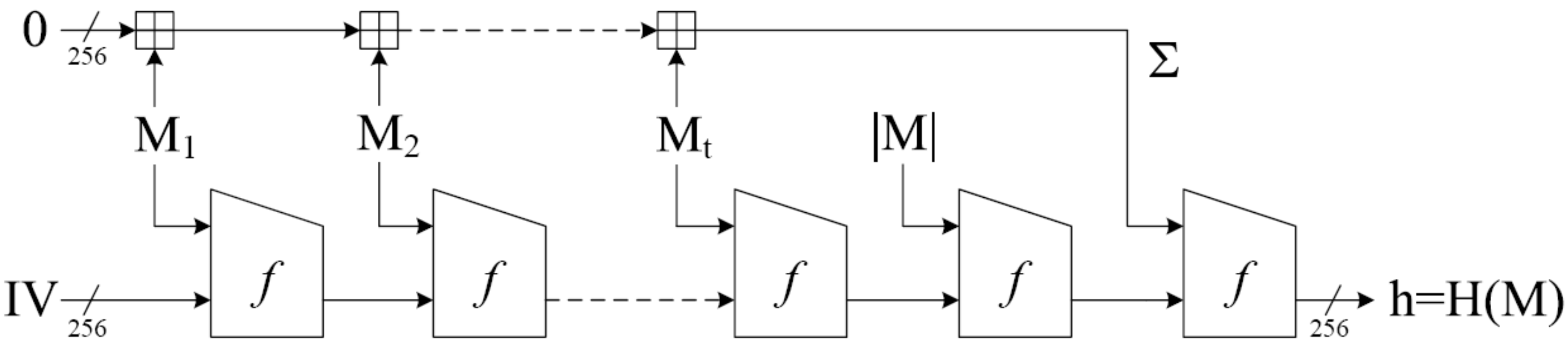
- Preimage resistance
  - Attack complexity should be 2^n
- Second-Preimage resistance
  - Attack complexity should be 2^n
- Collision resistance
  - Attack complexity should be $2^{n/2}$

Outline

- Motivation
- Description of GOST
- A preimage attack on the GOST hash function (FSE 2008)
  - A pseudo-preimage attack on the compression function
  - A preimage attack on the hash function
- Improving the attack (work in progress)
  - A fixed-point in the GOST block cipher
  - Improving the preimage attack on the hash function
  - A collision attack on the hash function
- Conclusion and Future Work

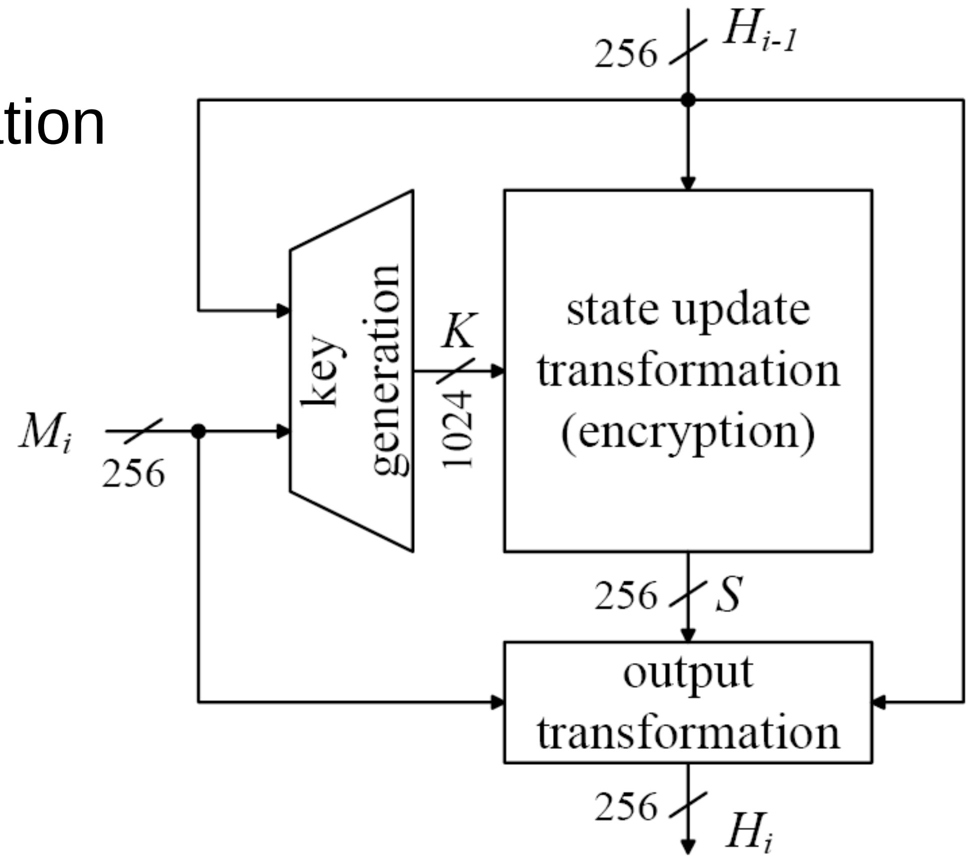
The GOST Hash Function

- The GOST hash function was published 1994
- Iterated Hash Function processes 256-bit blocks and produces a 256-bit hash value



The compression function of GOST

- The compression function of GOST consists of 3 parts
- State Update Transformation
- Key Generation
- Output Transformation

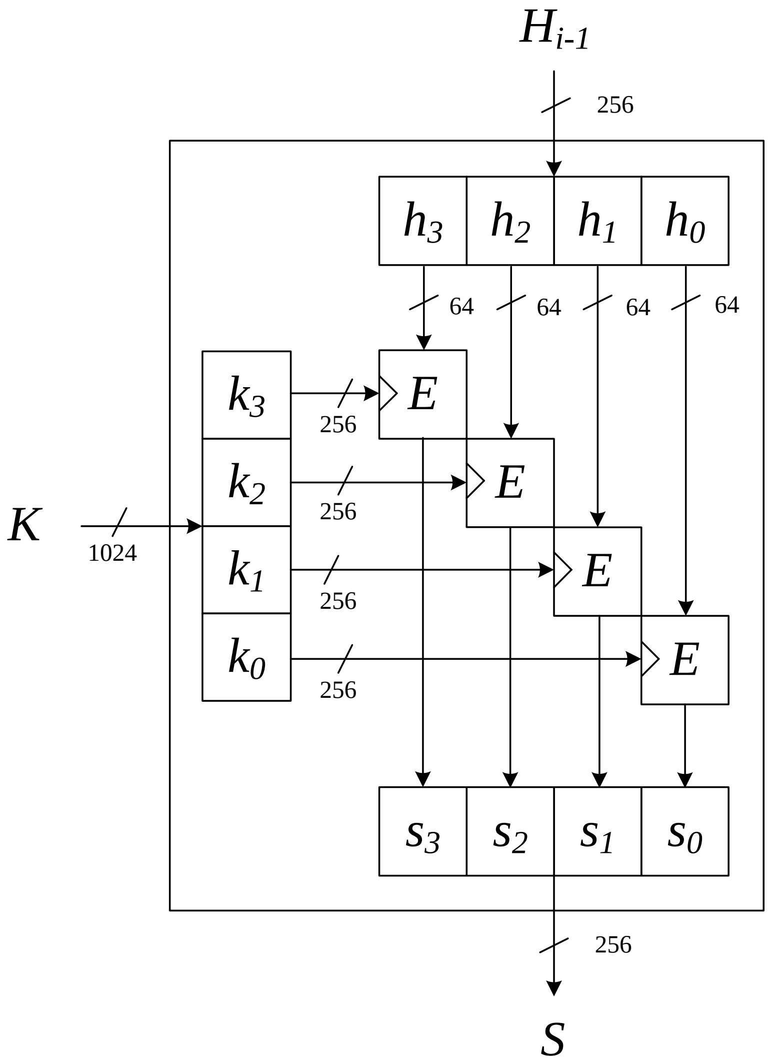


The State Update Transformation

Takes as input the intermediate hash value H_{i-1} and the key K to compute S

$$s_0 = E(k_0, h_0)$$
$$s_1 = E(k_1, h_1)$$
$$s_2 = E(k_2, h_2)$$
$$s_3 = E(k_3, h_3)$$

where E denotes an encryption with the GOST block cipher



The Key Generation

- Takes as input the intermediate hash value H_{i-1} and the message block M_i to compute the 1024-bit key K

$$k_0 = P(H_{i-1} \oplus M_i)$$
$$k_1 = P(A(H_{i-1}) \oplus A^2(M_i))$$
$$k_2 = P(A^2(H_{i-1}) \oplus \texttt{Const} \oplus A^4(M_i))$$
$$k_3 = P(A(A^2(H_{i-1}) \oplus \texttt{Const}) \oplus A^6(M_i))$$

where A and P are linear transformations.

The Output Transformation

- The output transformation combines the intermediate hash value H_{i-1}, the message block M_i and the output of the stat update transformation S to compute the output H_i

$$H_i = \psi^{61}(H_{i-1} \oplus \psi(M_i \oplus \psi^{12}(S)))$$

The linear transformation $\psi : \{0,1\}^{256} \rightarrow \{0,1\}^{256}$ is given by

$$\psi(\Gamma) = (\gamma_0 \oplus \gamma_1 \oplus \gamma_2 \oplus \gamma_3 \oplus \gamma_{12} \oplus \gamma_{15}) \| \gamma_{15} \| \gamma_{14} \| \cdots \| \gamma_1$$

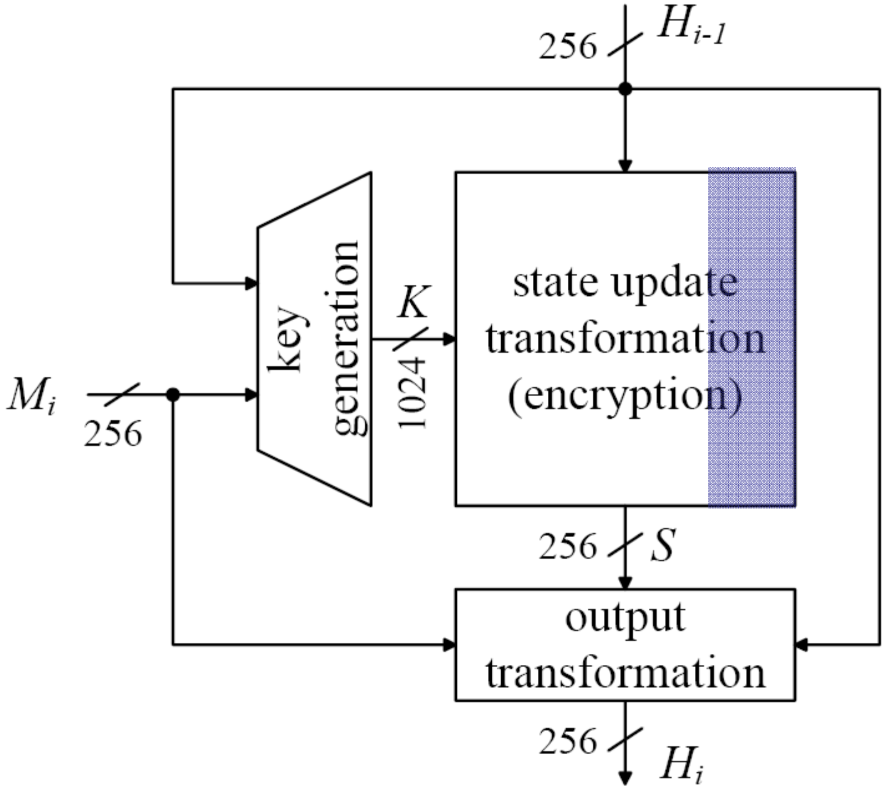
where $\Gamma = \gamma_{15} \| \gamma_{14} \| \cdots \| \gamma_0$

Outline

- Motivation
- Description of GOST
- A preimage attack on the GOST hash function (FSE 2008)
  - A pseudo-preimage attack on the compression function
  - A preimage attack on the hash function
- Improving the attack (work in progress)
  - A fixed-point in the GOST block cipher
  - Improving the preimage attack on the hash function
  - A collision attack on the hash function
- Conclusion and Future Work

Basic Attack Strategy

- Construct pairs (H_{i-1}, M_i) where parts of S (64 bits) are equal
- If we can construct these pairs efficiently then we can construct a pseudo-preimage with a complexity of about 2^{192}



Pseudo-Preimage for the Compression Function

- Since the output transformation of GOST

$$H_i = \psi^{61}(H_{i-1} \oplus \psi(M_i \oplus \psi^{12}(S)))$$

is linear, it can be also written as

$$H_i = \psi^{61}(H_{i-1}) \oplus \psi^{62}(M_i) \oplus \psi^{74}(S)$$

- Furthermore, ψ is invertible and hence, can be written as

$$\underbrace{\psi^{-74}(H_i)}_{X} = \underbrace{\psi^{-13}(H_{i-1})}_{Y} \oplus \underbrace{\psi^{-12}(M_i)}_{Z} \oplus S$$

Pseudo-Preimage for the Compression Function

- Split the words X,Y,Z into 64-bit words

$$X = x_3 \| x_2 \| x_1 \| x_0 \quad Y = y_3 \| y_2 \| y_1 \| y_0 \quad Z = z_3 \| z_2 \| z_1 \| z_0$$

then the previous equation can be written as:

$$x_0 = y_0 \oplus z_0 \oplus s_0 \quad \Leftarrow \quad s_0 = E(k_0, h_0)$$

$$x_1 = y_1 \oplus z_1 \oplus s_1$$

$$x_2 = y_2 \oplus z_2 \oplus s_2$$

$$x_3 = y_3 \oplus z_3 \oplus s_3$$

Pseudo-Preimage for the Compression Function

- We want to construct pairs (H_{i-1}, M_i) where $s_0 = E(k_0, h_0)$ is equal for each pair
- k_0 depends linearly on H_{i-1} and M_i : $P(H_{i-1} \oplus M_i)$
- To keep s_0 constant the following equations have to be fulfilled

$$\left.\begin{aligned} h_0 &= a \\ m_0 \oplus h_0 &= b_0 \\ m_1 \oplus h_1 &= b_1 \\ m_2 \oplus h_2 &= b_2 \\ m_3 \oplus h_3 &= b_3 \end{aligned}\right\} \text{arbitrary}$$

Pseudo-Preimage for the Compression Function

- Once we have fixed k_0 and h_0 and hence s_0, we have to fix y_0 and z_0 to guarantee that

$$x_0 = y_0 \oplus z_0 \oplus s_0$$

is correct with $X = x_3 \| x_2 \| x_1 \| x_0$ and $X = \psi^{-74}(H_i)$

- This adds the following equation

$$x_0 \oplus z_0 = c = x_0 \oplus s_0$$

to the our system of equations over $GF(2)$

Pseudo-Preimage for the Compression Function

- In total we get a system of 6*64 linear equations in 8*64 variables over *GF(2)*

$$
\begin{aligned}
y_0 \oplus z_0 &= c \\
h_0 &= a \\
m_0 \oplus h_0 &= b_0 \\
m_1 \oplus h_1 &= b_1 \\
m_2 \oplus h_2 &= b_2 \\
m_3 \oplus h_3 &= b_3
\end{aligned}
$$

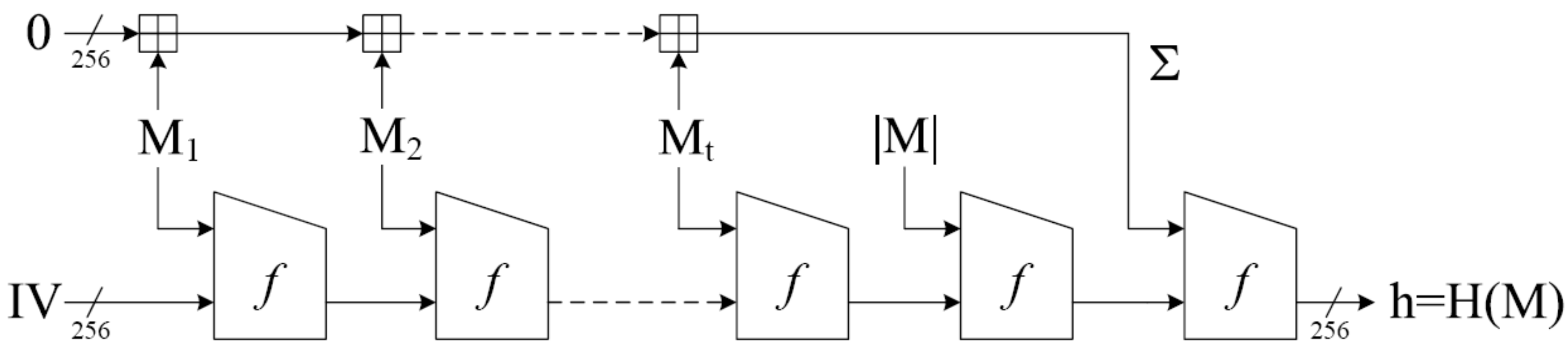
- We use this to construct a pseudo-preimage for the compression function of GOST

Pseudo-Preimage for the Compression Function

- By solving this system of equations over $GF(2)$ we get 2^{128} pairs (H_{i-1}, M_i), where x_0 is correct.
- For each pair compute X and check if x_1, x_2, x_3 are correct
- After testing all 2^{128} pairs we will find a correct pair with probability 2^{-64}
- By repeating the attack about 2^{64} times (with different values for a, b_0, b_1, b_2, b_3) we will find a pseudo-preimage for the compression function of GOST
- Constructing a pseudo-preimage has a complexity of 2^{192}

A Preimage for the Hash Function

How can we turn the pseudo-preimage attack on the compression function into a preimage attack on the hash function?

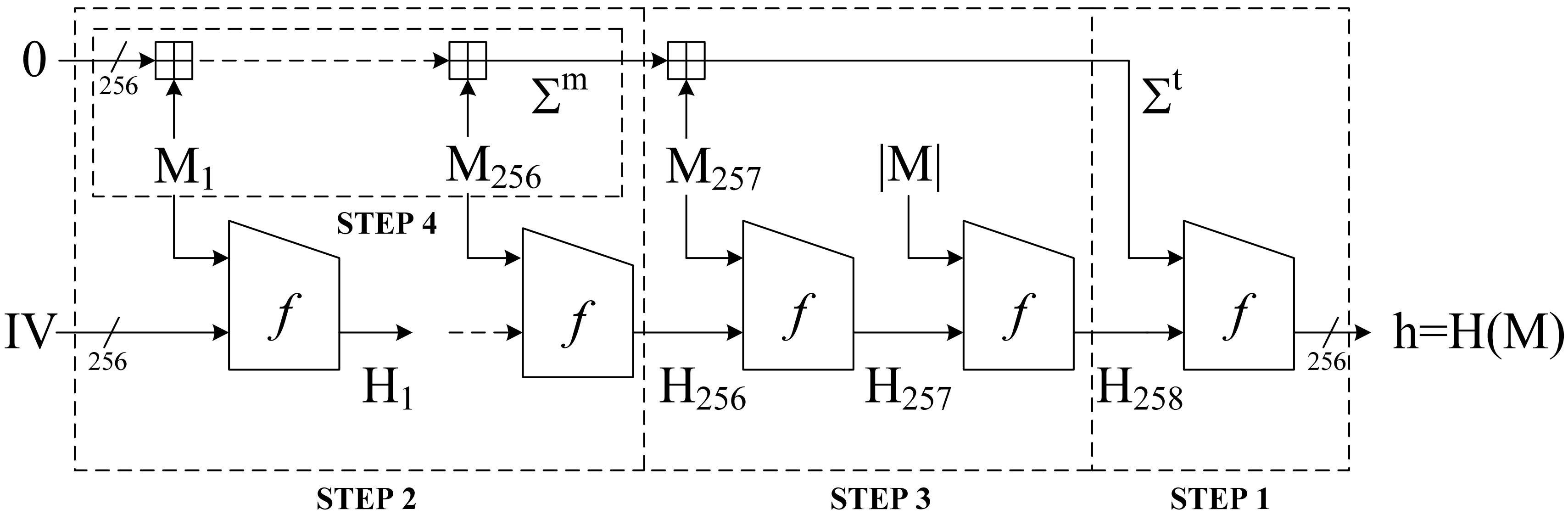


Problems:

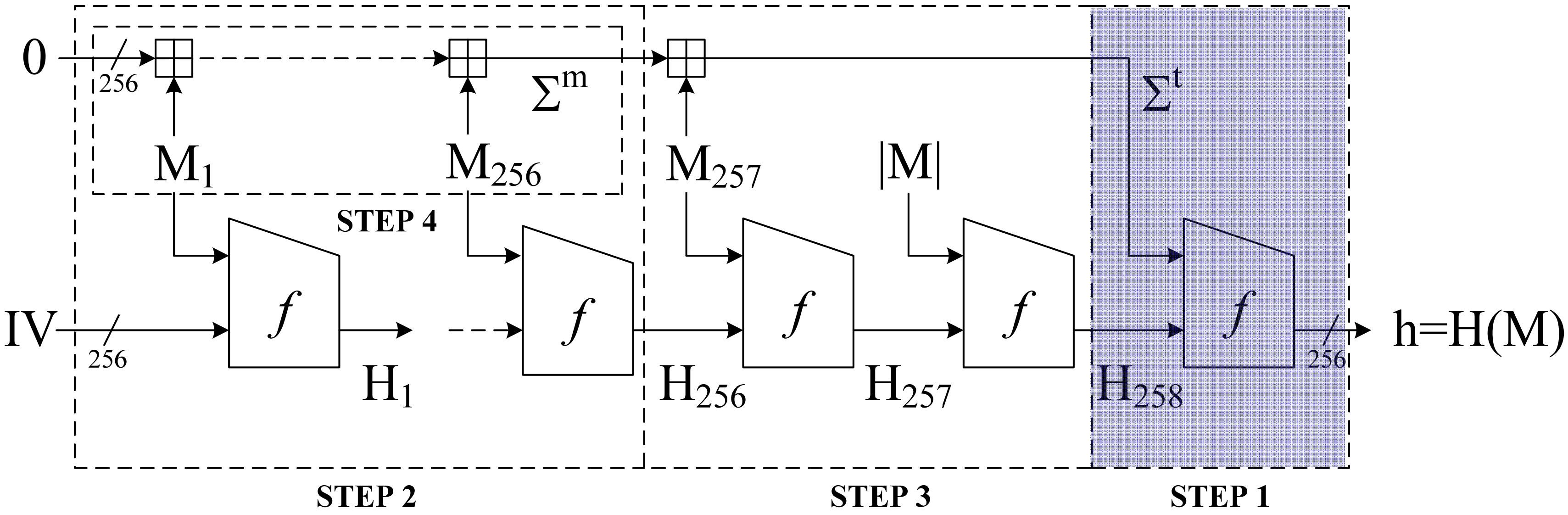
- Checksum over all message words
- Padding

Outline of the Attack

- Assume we want to construct a preimage for GOST consisting of 257 message blocks
- The attack basically consist of 4 steps

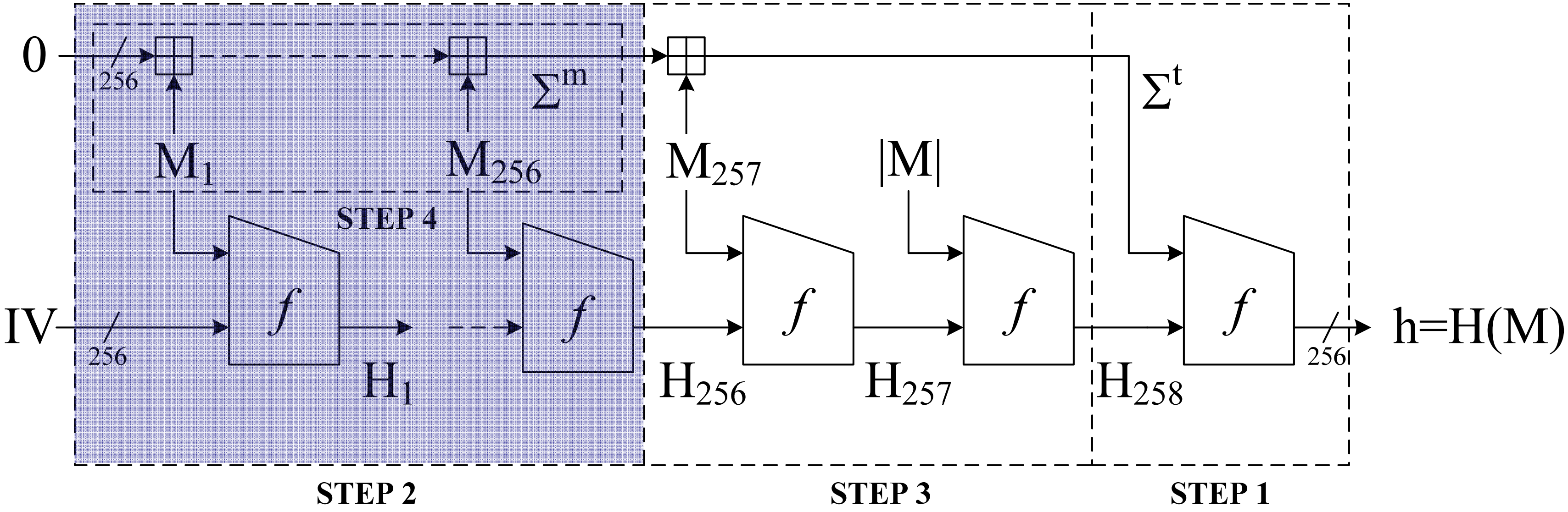


Outline of the Attack



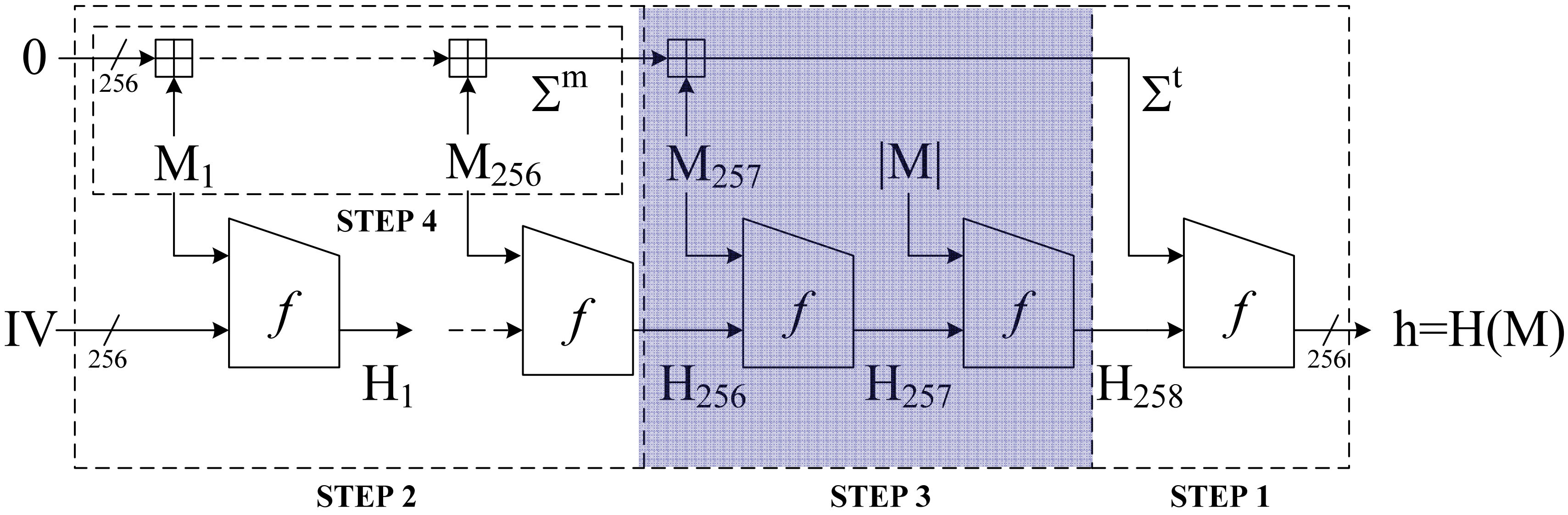
- **STEP 1**
  - Construct 2^{32} pseudo-preimages for the last iteration of GOST and save the 2^{32} pairs (H_{258}, Σ^t) in the list L
  - This has a complexity of about 2^{224} evaluations of the compression function of GOST

Outline of the Attack



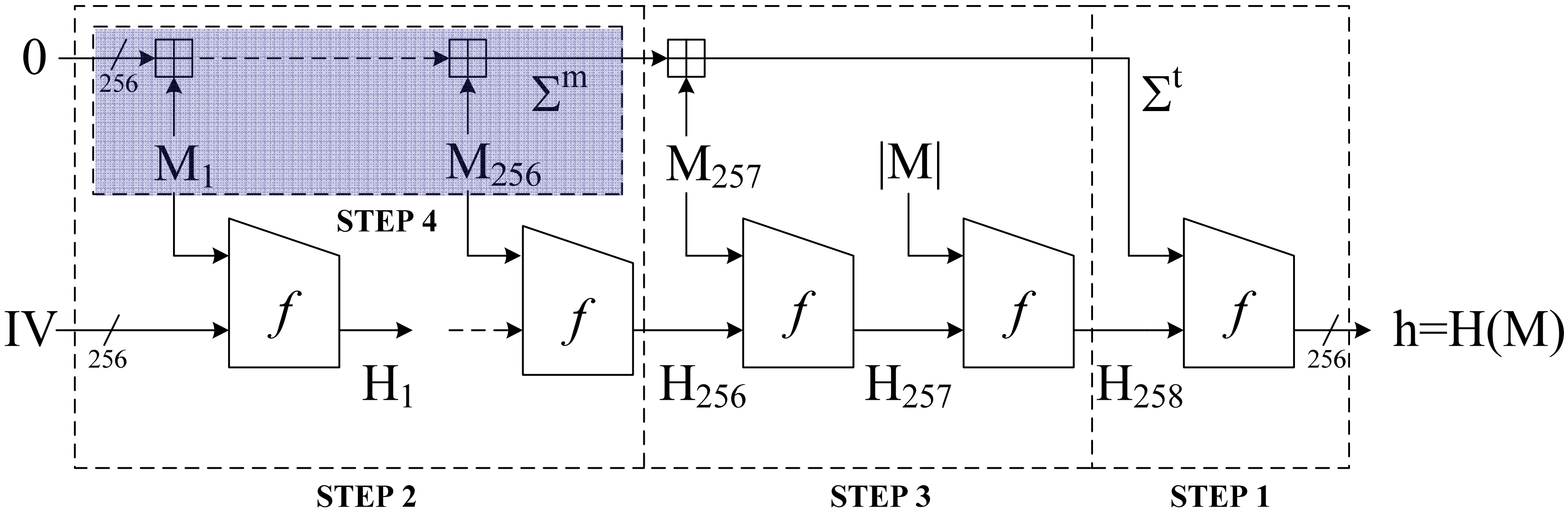
- **STEP 2**
  - Construct a 2^{256} multicollision for the first 256 message blocks
  - Thus, we have 2^{256} messages $M^* = M_1^{j_1} \| M_2^{j_2} \| \cdots \| M_{256}^{j_{256}}$ which all lead to the same intermediate hash value H_{256}.

Outline of the Attack



- **STEP 3**
  - Find a message block M_{257} such that for the given H_{256} and $|M|$ we find a H_{258} which is also contained in the list L
  - This has a complexity of about 2^{225} evaluations of the compression function of GOST

Outline of the Attack



- **STEP 4**
  - From the set of 2^{256} messages $M^* = M_1^{j_1} \| M_2^{j_2} \| \cdots \| M_{256}^{j_{256}}$ find a message that lead to $\Sigma^m = \Sigma^t \boxminus M_{257}$
  - This can be done by applying a meet-in-the-middle approach

STEP 4: Constructing the needed value in the checksum

- From a set of 2^{256} messages $M^* = M_1^{j_1} \| M_2^{j_2} \| \cdots \| M_{256}^{j_{256}}$ we have to find a message M^* that leads to the needed value Σ^m

Outline of the attack:

- Save all 2^{128} values for $\Sigma_1 = M_1^{j_1} \boxplus M_2^{j_2} \boxplus \cdots \boxplus M_{128}^{j_{128}}$ in the list L
- For all values $\Sigma_2 = M_{129}^{j_{129}} \boxplus M_{130}^{j_{130}} \boxplus \cdots \boxplus M_{256}^{j_{256}}$ check if there is a entry in the list L

After testing at most 2^{128} values we expect to find a matching entry in the list L

Complexity of the Attack

- The complexity of the attack is dominated by **STEP 1** and **STEP 3** of the attack.

STEP 1	STEP 2	STEP 3	STEP 4
2^{224}	2^{137}	2^{225}	2^{129}

- Note that a memory less variant of the meet-in-the-middle attack can be used in **STEP 4** of the attack to reduce the memory requirements.

STEP 1	STEP 2	STEP 3	STEP 4
2^{38}	2^{13}	-	-

Summary

- We have shown a pseudo-preimage attack on the compression function of GOST with a complexity of 2^{192}
- We have shown a preimage attack on the GOST hash function with a complexity of about 2^{225} and memory requirements of 2^{38} bytes
- Both attacks are *independent* of the GOST block cipher

Improving the Attack

Florian Mendel, Norbert Pramstaller, Christian Rechberger,
Marcin Kontak, and Janusz Szmidt

(work in progress)

Institute for Applied Information Processing and Communications (IAIK) - <u>Krypto Group</u>

Faculty of Computer Science
Graz University of Technology

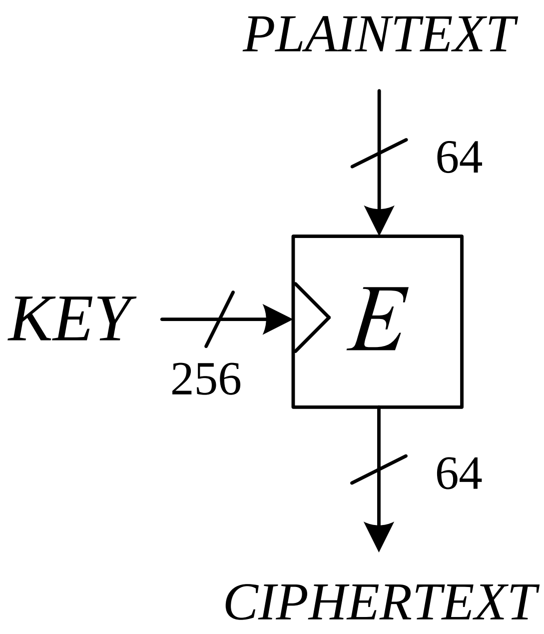
Outline

- Motivation
- Description of GOST
- A preimage attack on the GOST hash function (FSE 2008)
  - A pseudo-preimage attack on the compression function
  - A preimage attack on the hash function
- Improving the attack (work in progress)
  - A fixed-point in the GOST block cipher
  - Improving the preimage attack on the hash function
  - A collision attack on the hash function
- Conclusion and Future Work

The GOST Block Cipher

- The block cipher was published in 1989
- It is Russian government standard (GOST 28147-89)

- Block size: 64 bits
- Key size of 256 bits

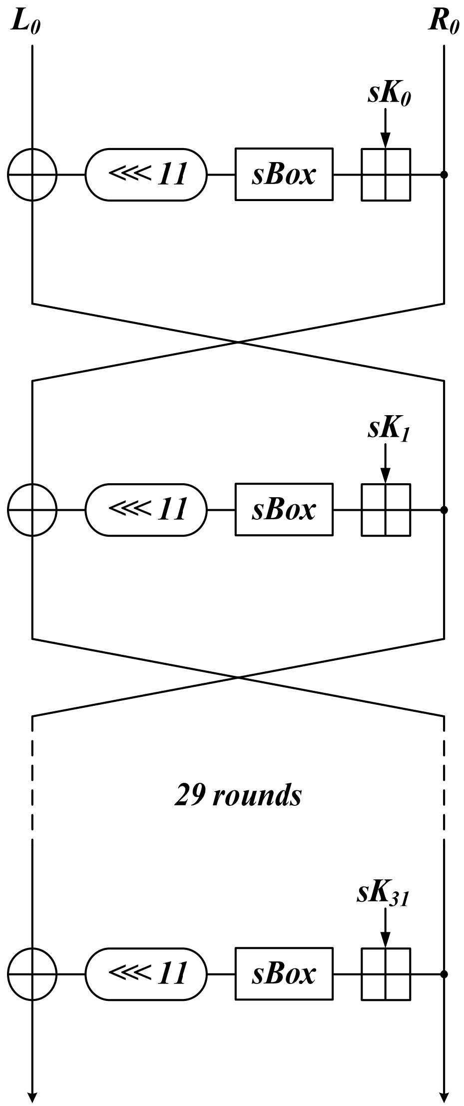


The GOST Block Cipher

- 32-round Feistel network
- 32-bit round key sK_i
- Simple Key Schedule

$$sK_i = \begin{cases} K_i \pmod 8, & \text{if } i < 24, \\ K_{7-i} \pmod 8 & \text{otherwise} \end{cases}$$

where $K = K_7 \| \cdots \| K_0$



A fixed-point in the GOST Block Cipher

- If we can construct a fixed-point in the first 8 rounds of GOST, then we have also a fixed-point for 32 round if $L_0=R_0$
- Construction a fixed-point in the first 8 rounds is easy, since each word of the key is only used once
- Example: $L_0= R_0= 0$

1	34A451AA	C2DF23D8	CAE90664	C3965FE0	E9298408	C5987119	5FC1DA30	B1A5E9DB
2	FF48B08A	3C3812E3	8F78C57E	9742312A	769D919D	D269902E	5782AEAF	B515779F
3	31E4AE5A	E8FB14D0	47B5C0F1	3F07C5D9	CA156F01	CE2BEABB	A20F384D	26776A13
4	A8E418B8	6C759B2C	ADE4574B	B7F93FA1	D40E9A48	6D324773	43ECC12D	CBC28A89
5	1F27086C	524D5E31	07395FDE	7056AF86	D26A644D	D2F37938	3D0BF7DE	C5109C6D
6	4067D0F7	D31A7AD9	0DA7E0C2	F4975099	285C0267	8CB7C0A6	8A7FA3EC	62A65517
7	A222761E	6DB7D3CB	4A6C316B	65103CC8	75402050	DEB5DDC5	E470253F	487334D5
8	0ABDC454	18E4D226	6BDDFCC4	C477B694	5D39FB39	E50480CC	B074FF43	3B388231

Improving the Preimage Attack on the Hash Function

- We want to construct many message blocks M_i where $s_0 = E(k_0 , h_0)$ is equal for a fixed value of H_{i-1} ($h_0 = 0$)

$$x_0 = y_0 \oplus z_0 \oplus s_0 \qquad \Leftarrow \qquad x_0 = y_0 \oplus z_0 \oplus h_0$$

$$x_1 = y_1 \oplus z_1 \oplus s_1$$

$$x_2 = y_2 \oplus z_2 \oplus s_2$$

$$x_3 = y_3 \oplus z_3 \oplus s_3$$

- Now x_0 depends linearly on H_{i-1} and M_i, to guarantee that x_0 is correct the following equation has to be fulfilled:

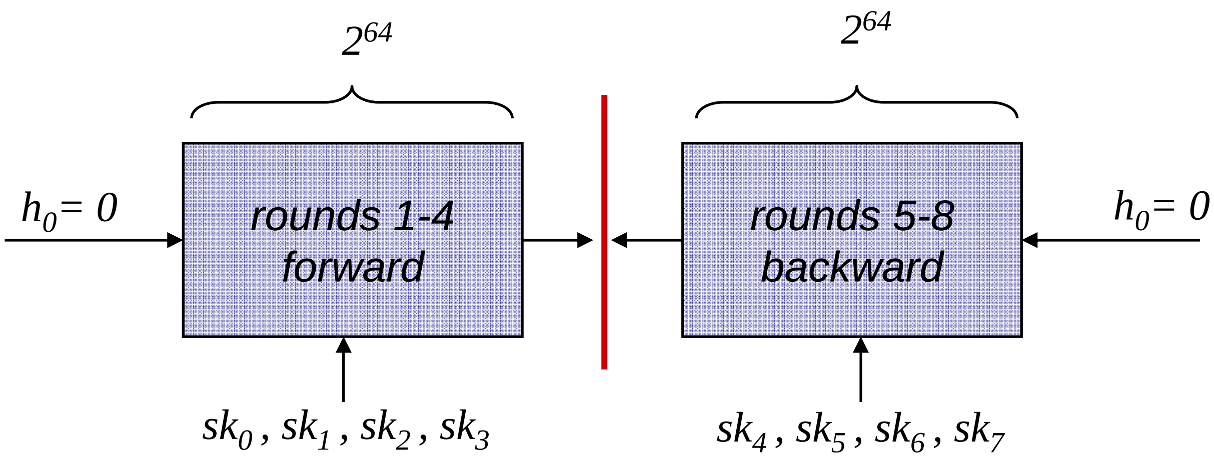
$$z_0 = x_0 \oplus y_0 \oplus h_0$$

Improving the Preimage Attack on the Hash Function

- Since z_0 depends linearly on M_i this restricts our choices of the key $k_0 = P(H_{i-1} \oplus M_i)$
- Hence, constructing a fixed-point for $s_0 = E(k_0, h_0)$ gets more complicated
- But still:
  - To construct a fixed-point in the GOST block cipher we only need to construct a fixed-point in the first 8 rounds if $h_0 = 0$
  - In the first 8 rounds each word of the key is only used once

Constructing many fixed-points

- We use a meet in the middle attack to construct 2^{64} fixed-points with a complexity of about 2^{64} evaluations of the GOST block cipher



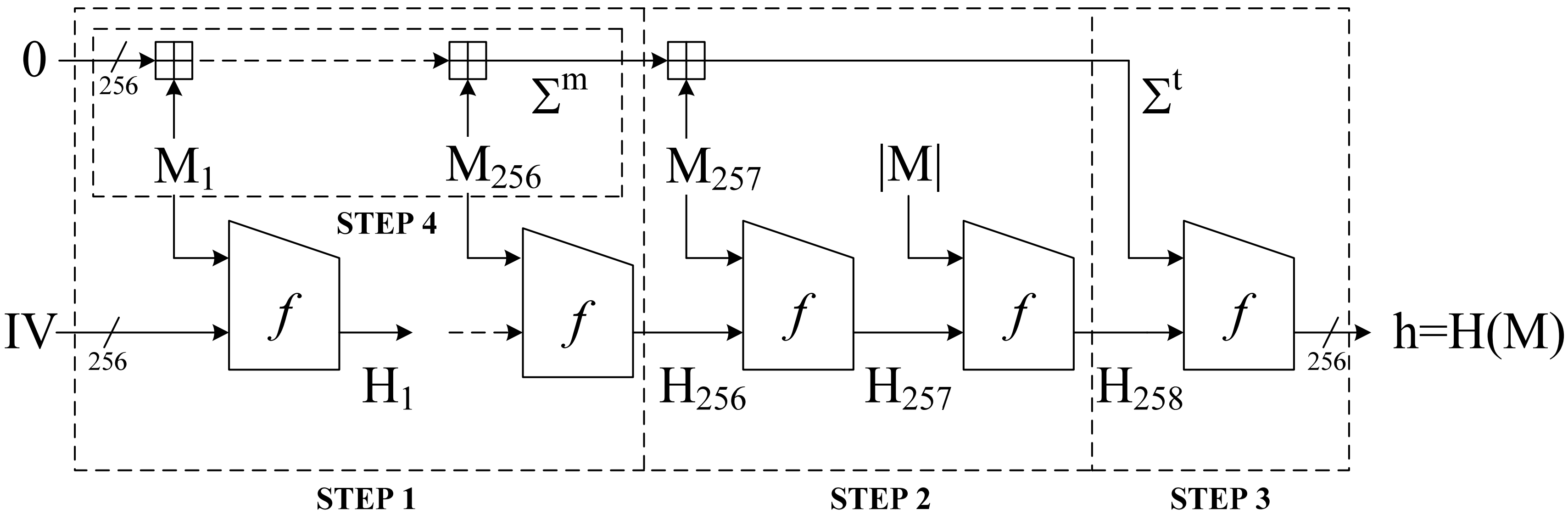
Note that the choice of the 8 subkeys $sk_0,\ldots,sk_7$ is restricted by 64 equations over $GF(2)$

Improving the Preimage Attack on the Hash Function

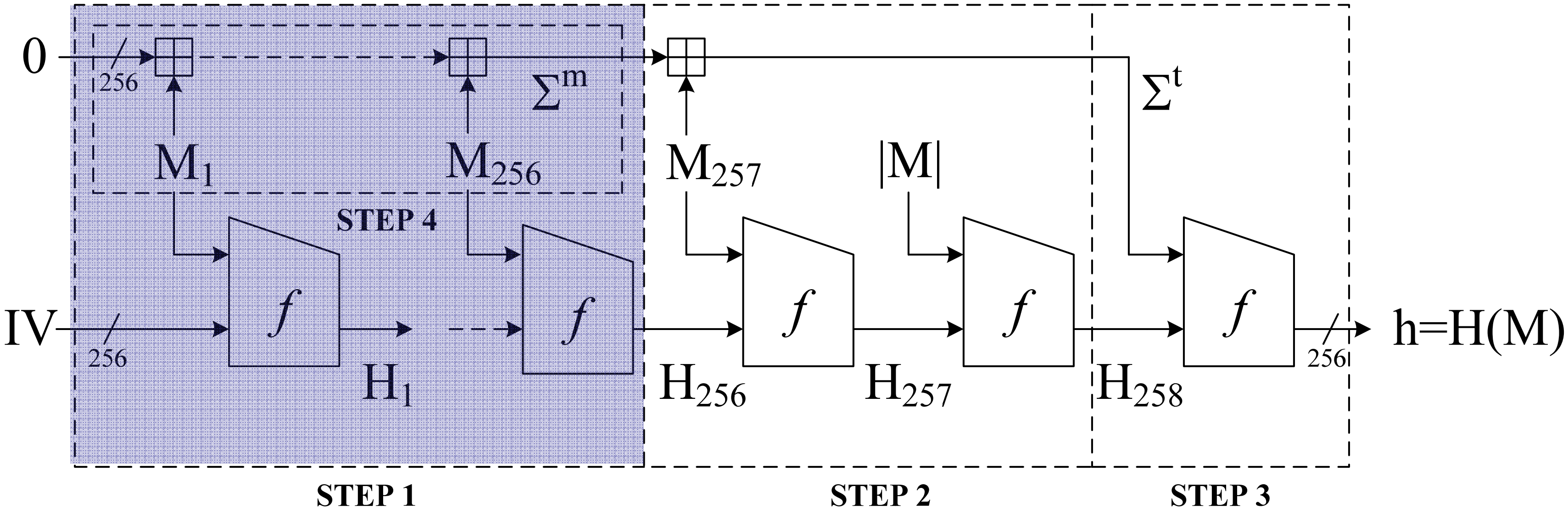
- With this method we get 2^{64} message blocks M_i , where x_0 is correct.
- We can repeat the attack 2^{64} times to construct 2^{128} message blocks M_i , where x_0 is correct.
- For each message block we compute X and check if x_1, x_2, x_3 are correct
- After testing all 2^{128} message blocks we will find a correct message block with probability 2^{-64}

Outline of the Attack

- Again assume we want to construct a preimage for GOST consisting of 257 message blocks
- The attack basically consist of 4 steps

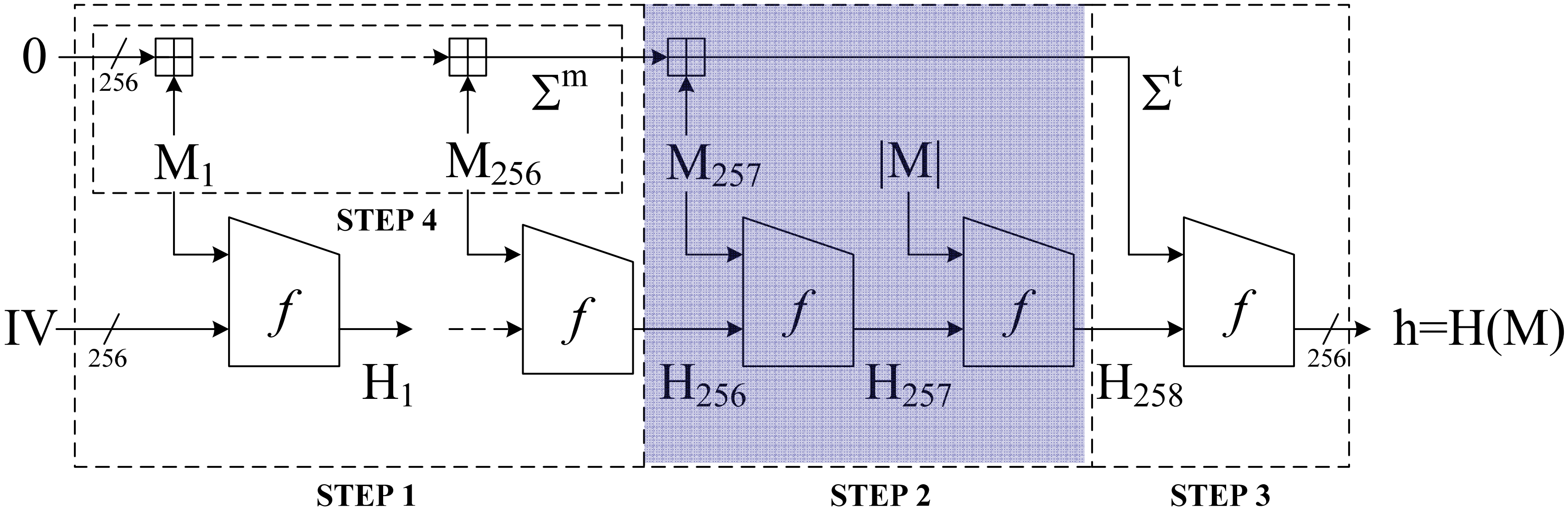


Outline of the Attack



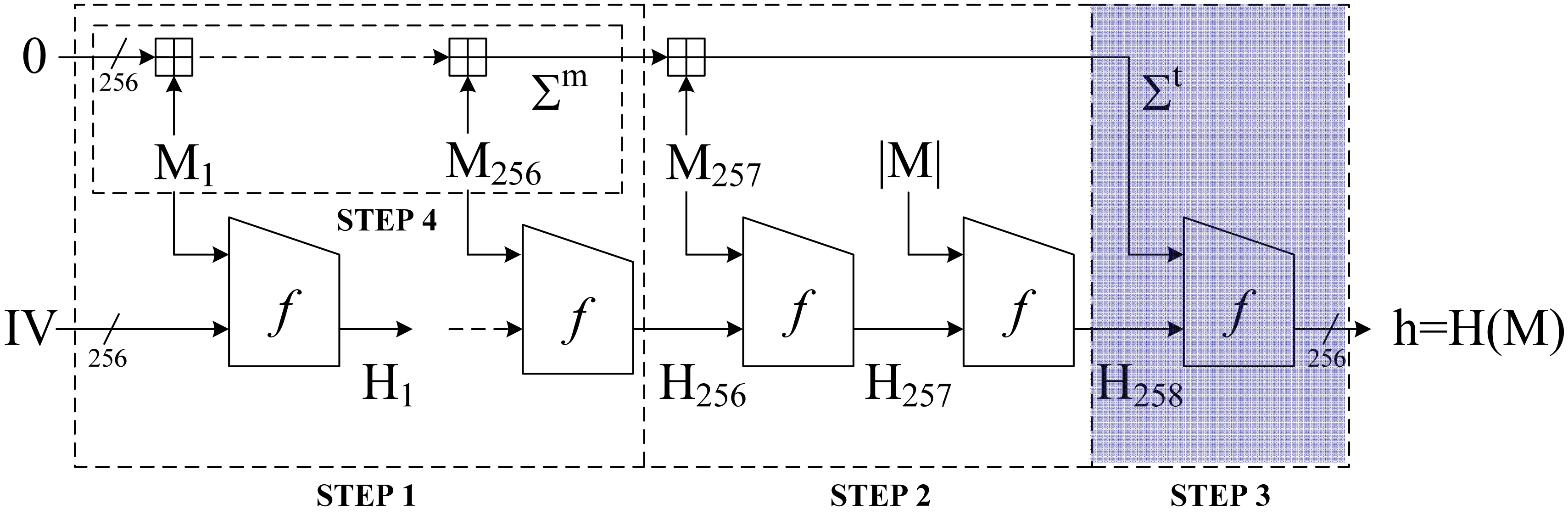
- **STEP 1**
  - Construct a 2^{256} multicollision for the first 256 message blocks
  - Thus, we have 2^{256} messages $M^* = M_1^{j_1} \| M_2^{j_2} \| \cdots \| M_{256}^{j_{256}}$ which all lead to the same intermediate hash value H_{256}.

Outline of the Attack



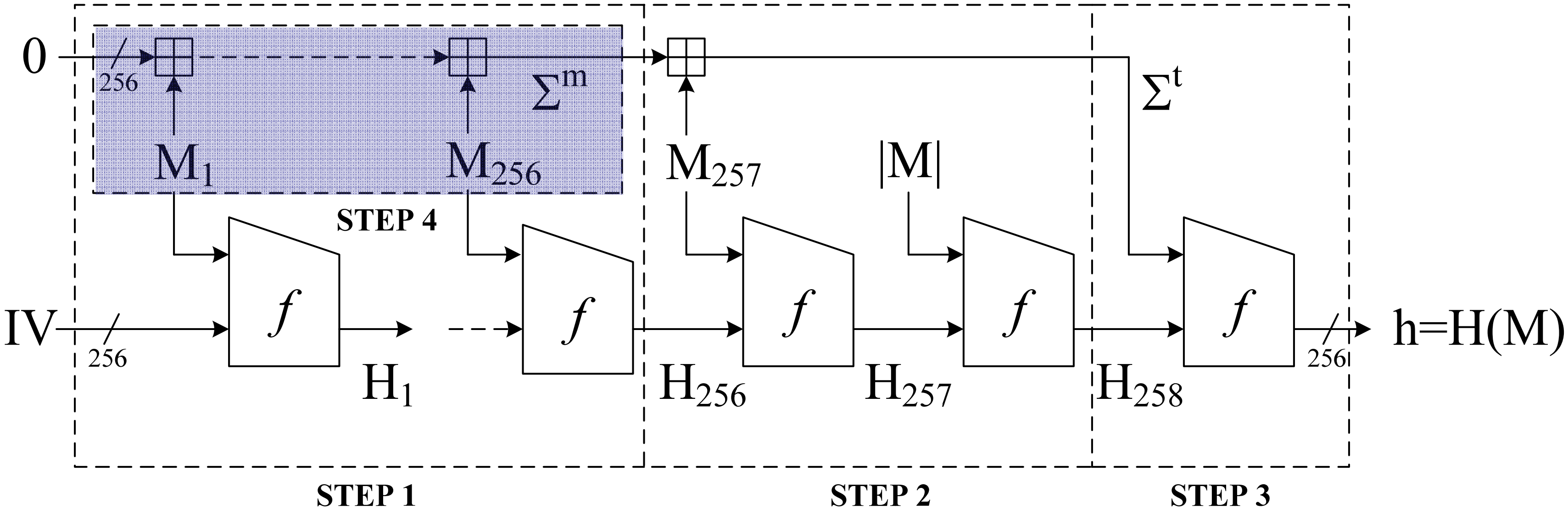
- **STEP 2**
  - Find a message block M_{257} such that for the given H_{256} and $|M|$ we find a H_{258} with $h_0 = 0$
  - This has a complexity of about 2^{64} evaluations of the compression function of GOST

Outline of the Attack



- **STEP 3**
  - Construct a preimage for the last iteration of the GOST hash function by constructing 2^{128} fixed-points (probability 2^{-64}).
  - If no preimage is found then go back to STEP 2

Outline of the Attack



- **STEP 4**
  - From the set of 2^{256} messages $M^* = M_1^{j_1} \| M_2^{j_2} \| \cdots \| M_{256}^{j_{256}}$ find a message that lead to $\Sigma^m = \Sigma^t \boxminus M_{257}$
  - This can be done by applying a meet-in-the-middle approach

Complexity of the Attack

- The complexity of the attack is dominated by **STEP 3** of the attack.

STEP 1	STEP 2	STEP 3	STEP 4
2^{137}	$2^{64} * 2^{64}$	$2^{64} * 2^{128}$	2^{129}

- The memory requirements of the attack are dominated by **STEP 3** of the attack

STEP 1	STEP 2	STEP 3	STEP 4
2^{13}	-	2^{69}	-

A Remark on Collision Attacks on GOST

- Since we can construct 2^{96} message blocks M_i which all produce the same x_0 we can construct a collision for the compression function (birthday attack)
- Again we can use multicollisions to turn the collision attack on the compression function into a collision attack on the hash function
- To construct also a collision in the checksum we use a generalized birthday attack to reduce the complexity of this step of the attack
- The collision attack has a complexity of about $2^{105} < 2^{128}$

Summary of Results

- By exploiting special properties of the compression function of GOST block Cipher we can construct preimages for the hash function with a complexity of about 2^{225} and memory requirements of 2^{38} bytes

- By exploiting special properties of the GOST block cipher we can find
  - Preimages for the GOST hash function with a complexity of about 2^{192} and memory requirements of 2^{69} bytes
  - Collisions GOST hash function with a complexity of about 2^{105} and memory requirements of 2^{69} bytes

Thank you for your Attention