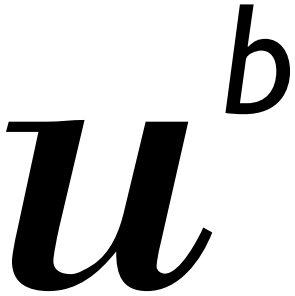


b

UNIVERSITÄT
BERN

UNIVERSITÄT BERN

Network Security

IV. Random Numbers and Hashing

Prof. Dr. Torsten Braun, Institut für Informatik

Bern, 14.03.2022 – 21.03.2022

UNIVERSITÄT BERN

Random Numbers and Hashing

Table of Contents

1. Random Numbers

1. Use of Random Numbers

- Key distribution (vs nonces)
- Session key generation
- Public-key encryption
- Symmetric stream encryption

1. Random Numbers

2.1 Randomness Criteria

Uniform distribution

– Frequency of occurrence of bits should be approximately equal.

Independence

– No subsequence can be inferred from another one.

1. Random Numbers

2.2 Unpredictability Criteria

Forward Unpredictability

– If seed is unknown, next output should be unpredictable.

Backward Unpredictability

– Seed should not be predictable from any output value.

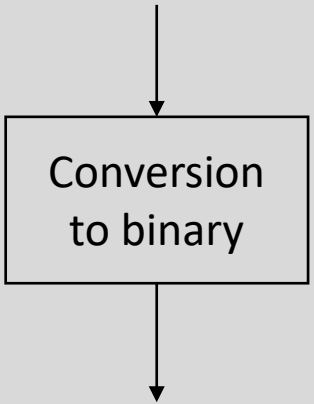
UNIVERSITÄT BERN

1. Random Numbers

3. Random Number Generation

True Random Number Generator

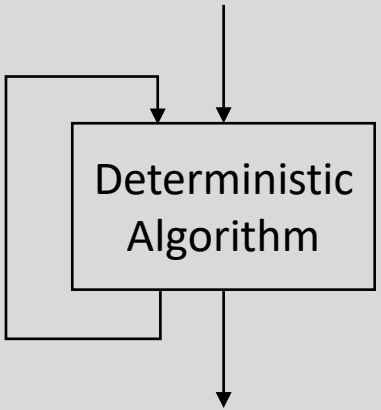
Source of true randomness



Random bit stream

Pseudorandom Number Generator

Seed

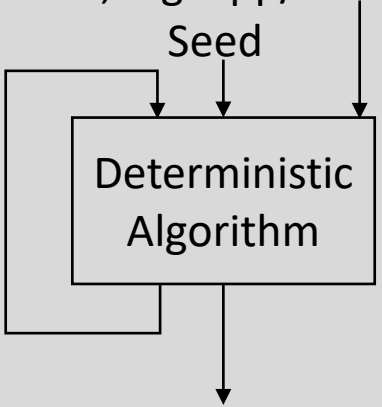


Pseudorandom bit stream

Pseudorandom Function

Context-specific value, e.g. app/user ID

Seed



Pseudorandom value

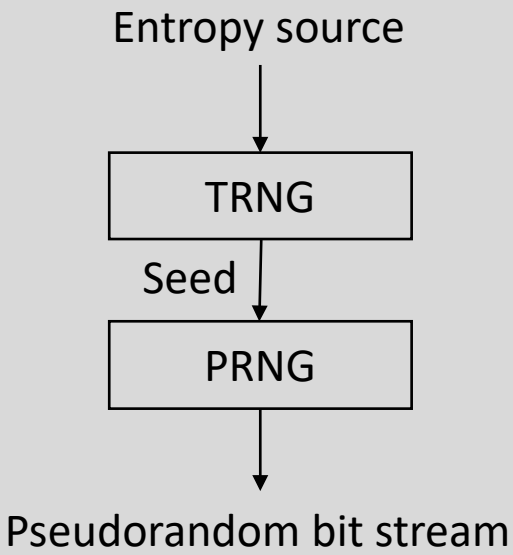
1. Random Numbers

4.1 TRNG: Entropy Sources

- Sound / video input
- Thermal noise
- Disk drives with random fluctuations in rotational speed
- Clock time

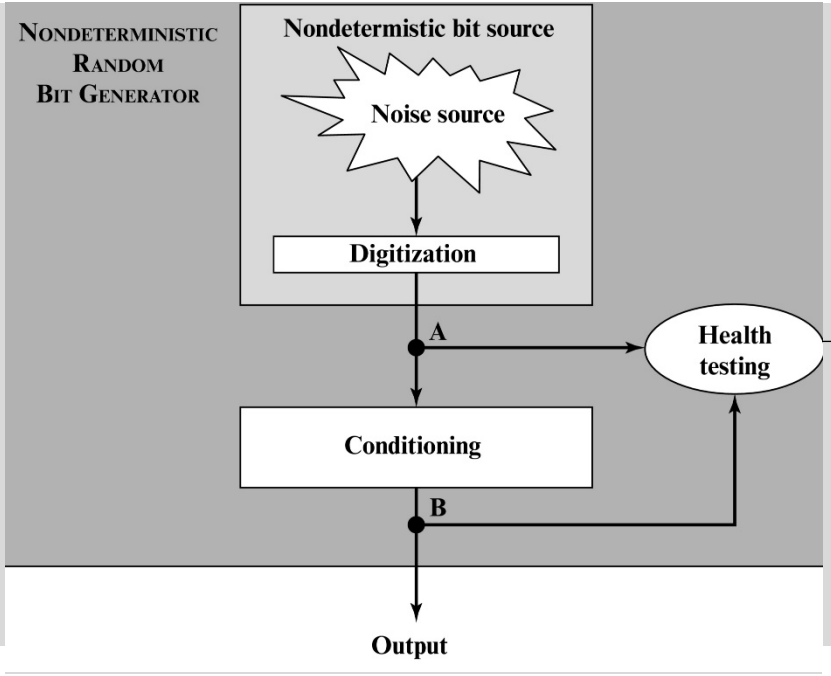
1. Random Numbers

4.2 TRNG as Seed Input Generation for PRNG



1. Random Numbers

4.3 Non-Deterministic Random Bit Generator



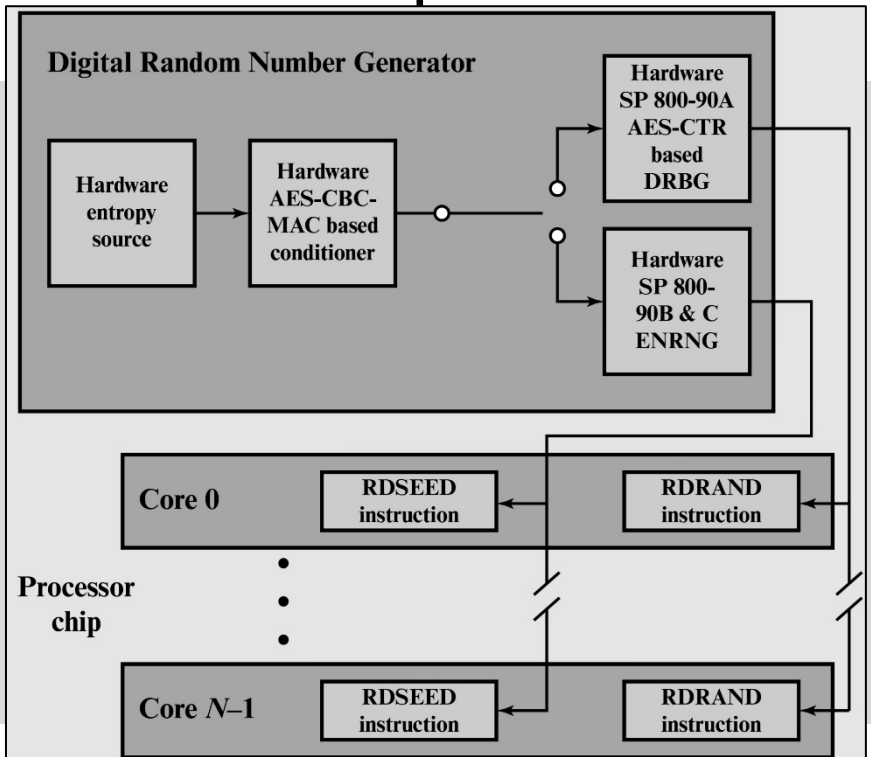
Test for variability, e.g.,

- repetition count test
- adaptive proportion test

UNIVERSITÄT BERN

1. Random Numbers

4.4 Intel Processor Chip with Random Number Generator



Deterministic Random Bit Generator

Enhanced Nondeterministic Random Number Generator

1. Random Numbers

5.1 PRNG: Linear Congruential Generators

- m: modulus, e.g., $2^{31}-1$
- a: multiplier
- c: increment
- X_0: seed

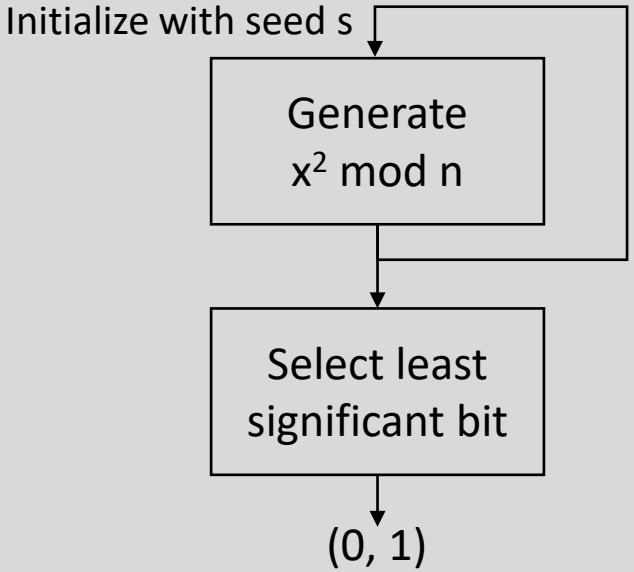
- $X_{n+1} = (a \cdot X_n + c) \bmod m$

Requirements for RNGs

- The function should be a full-period generating function. That is, the function should generate all the numbers from 0 through m−1 before repeating.
- The generated sequence should appear random.
- The function should implement efficiently with 32-bit arithmetic.

1. Random Numbers

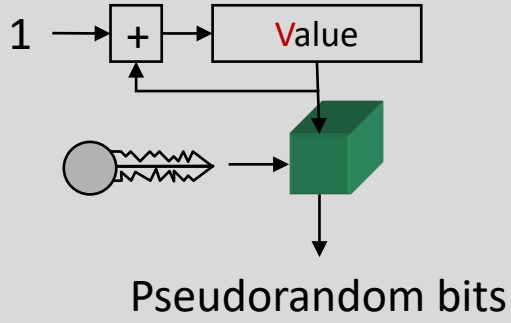
5.2 PRNG: Blum Blum Shub Generator



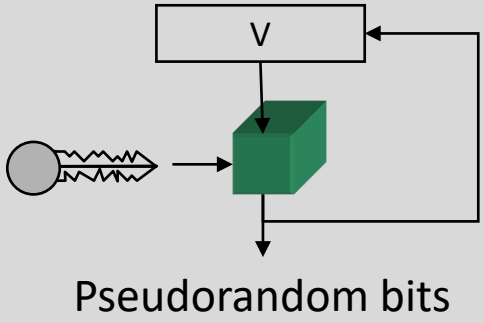
1. Random Numbers

5.3 PRNGs Using Block Cipher Modes of Operation

Counter Mode

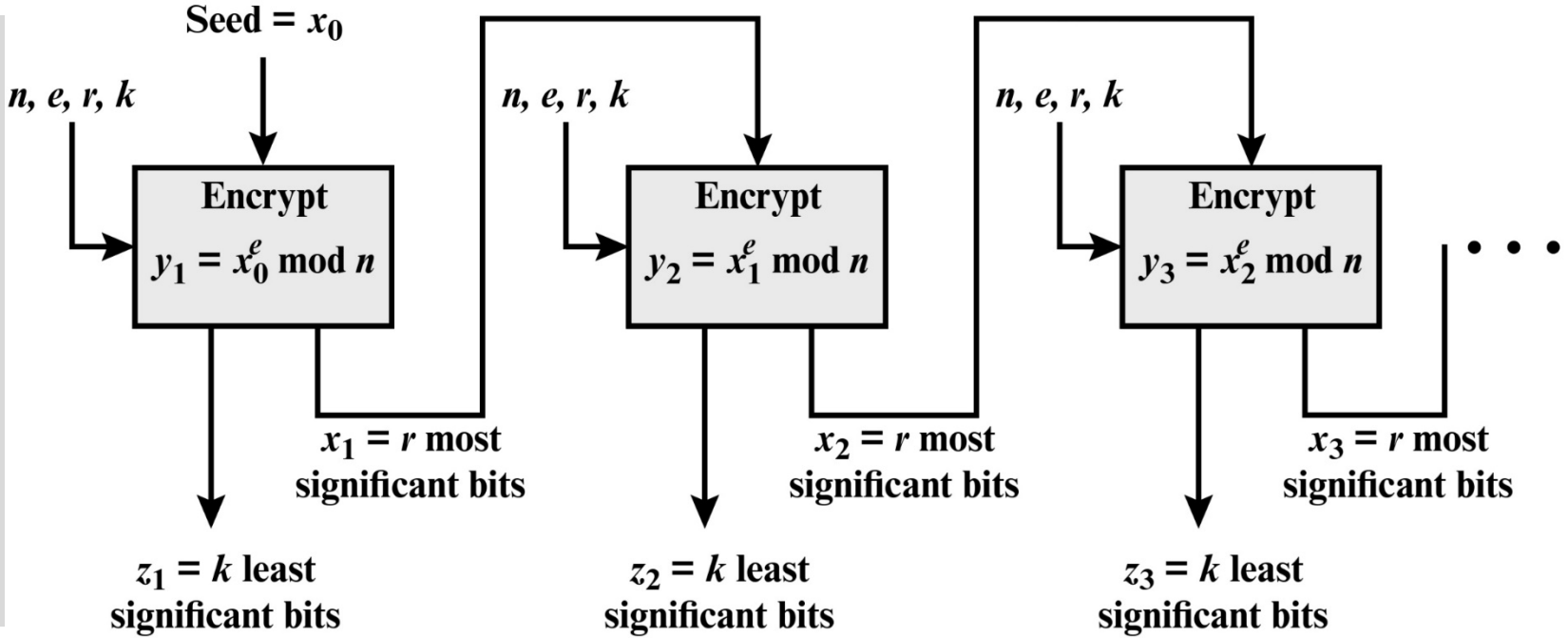


Output Feedback Mode



1. Random Numbers

5.4 PRNG based on RSA



1. Random Numbers

6. Comparison of PRNG and TRNG

PRNG

- Very efficient
- Deterministic
- Periodic

TRNG

- Generally inefficient
- Nondeterministic
- Aperiodic

2. Hashing

- Goal of (one-way) hashing function
  - Input: long message
  - Output: short block (called hash or message digest)
- Often combined with secret keys
- If possible: evenly distributed and random
- Example usage:
  - Calculation of fingerprints to detect modifications of data (digital signatures)
  - Message authentication

2. Hashing

1. Hashing using XOR

C_i: i^{th} bit of hash code

m: number of n-bit blocks in input

b_{ij}: i^{th} bit in j^{th} block

$\oplus$: XOR

$C_i = b_{i1} \oplus b_{i2} \oplus ... \oplus b_{im}$

- Effective for integrity checks, since each n-bit hash value is equally likely.
- Problem:
  more predictably formatted data, e.g., lower probability for highest bit in normal texts.

→ Rotation as another operation (rotated XOR)

- Possible encryption with CBC

2. Hashing

2.1 Cryptographic Hash Functions: Security Requirements

- Variable input sizes
- Fixed output size
- Efficiency (hardware and software implementations)

2. Hashing

2.2 Cryptographic Hash Functions: Security Requirements

- Pre-image resistant (one-way property)
  - For any given hash value h it is computationally infeasible to find a message M that produces h, i.e. finding M so that H(M) = h
- 2nd pre-image resistant (weak collision resistant)
  - for any given block M, it is computationally infeasible to find M', i.e. H(M) = H(M')
- (Strong) Collision resistant
  - It is computationally infeasible to find any pair (M, M') with M ≠ M' and H(M) = H(M').
- Pseudo-randomness
  - Output of H meets standard tests for pseudo-randomness

2. Hashing

2.3 Cryptographic Hash Functions: Brute-Force Attacks

Effort for brute-for-attacks for m-bit hash values

- 1st pre-image resistant: 2^m
- 2nd pre-image resistant: 2^m
- Collision resistant: $2^{m/2}$, cf. birthday paradox

2. Hashing

2.4 Cryptographic Hash Functions: Cryptanalysis

- focuses on internal structure of f
- tries to find efficient techniques to produce collisions of f

2. Hashing

2.5 Birthday Attack

- Alice wants to fire Fred.
- Bob wants to double Fred's salary.
- Alice signs firing letter.
  Bob writes double salary letter and substitutes signature.
- Bob needs to find two messages (letters) with the same message digest.

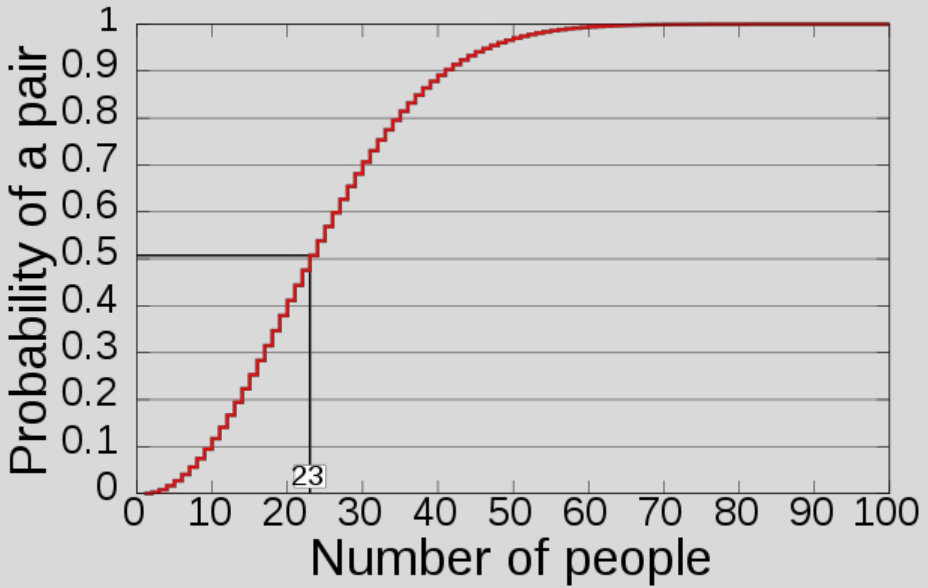
- Random variable 0...k-1
- Probability that a repeated element is encountered exceeds 0.5 after $\sqrt{(k-1)}$ choices.
  - n inputs: n (n-1) / 2 pairs of inputs
  - k outputs: probability of equal pair: 1/k
  - k/2 pairs for probability > 0.5
  - $n (n-1) / 2 > k / 2 \rightarrow n > \sqrt{k}$
- For an m-bit hash value, we can expect to find another message with the same hash value after $\sqrt{(2^m)} = 2^{m/2}$ attempts.

2. Hashing

2.6 Birthday Paradox

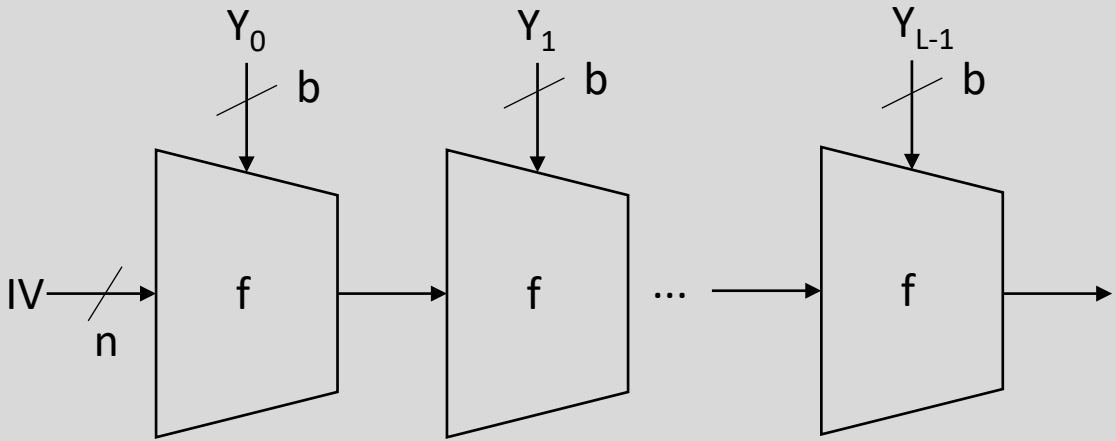
Example: How many persons needed to have a probability p > 0.5 that two persons share the same birthday ?

- 365^n different birthday combinations for n persons
- 365 · 364 · 363 · … · (365 – (n-1)) cases with different birthdays
- p = 1 – (365 · 364 · 363 · … · (365 – (n-1))) / 365^n



2. Hashing

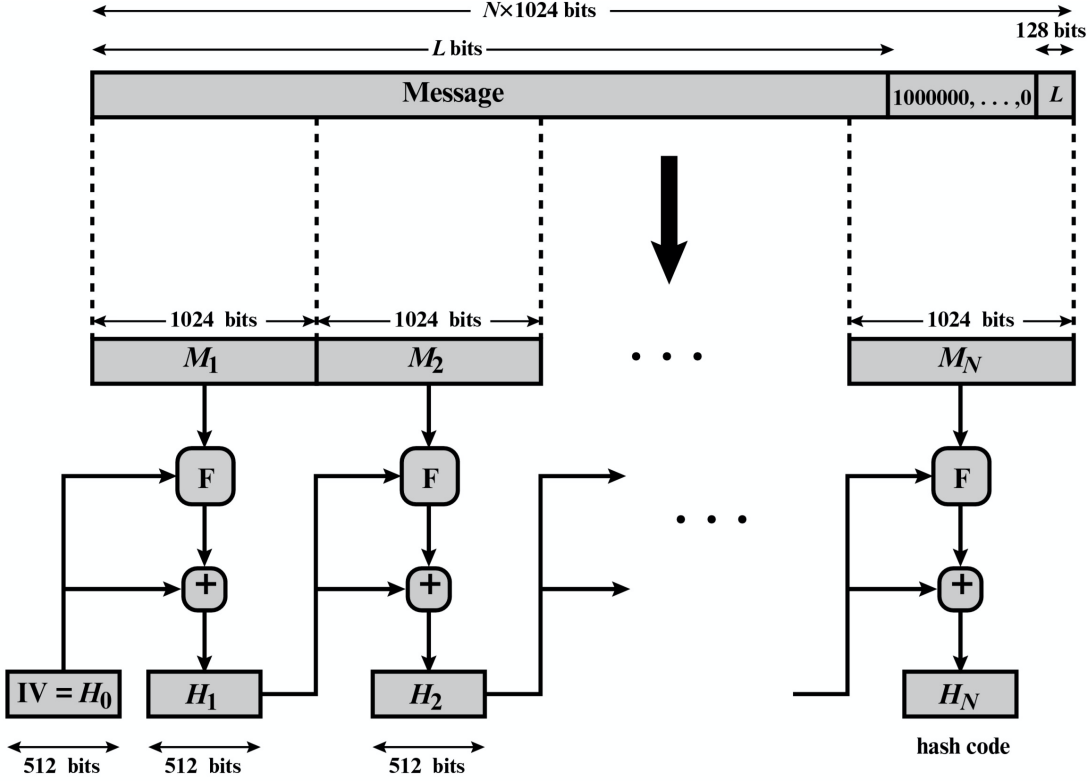
3.1 General Structure of Secure Hash Code



UNIVERSITÄT BERN

2. Hashing

3.2.1 SHA-512

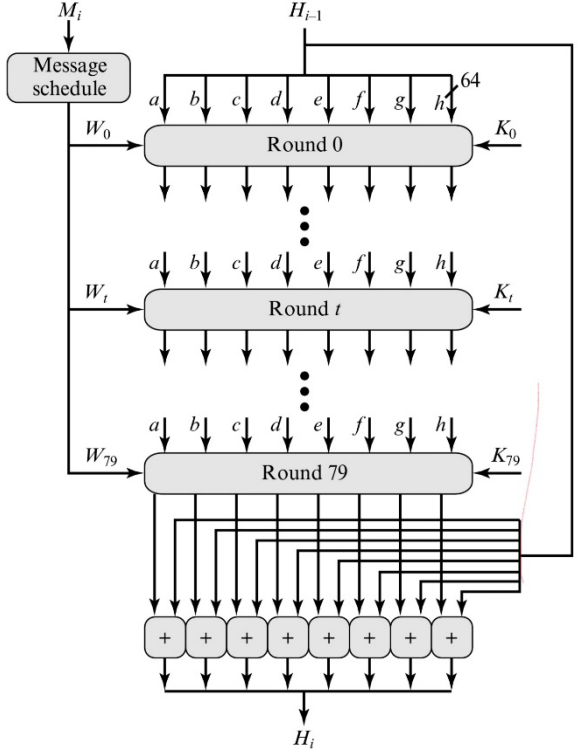


UNIVERSITÄT BERN

2. Hashing

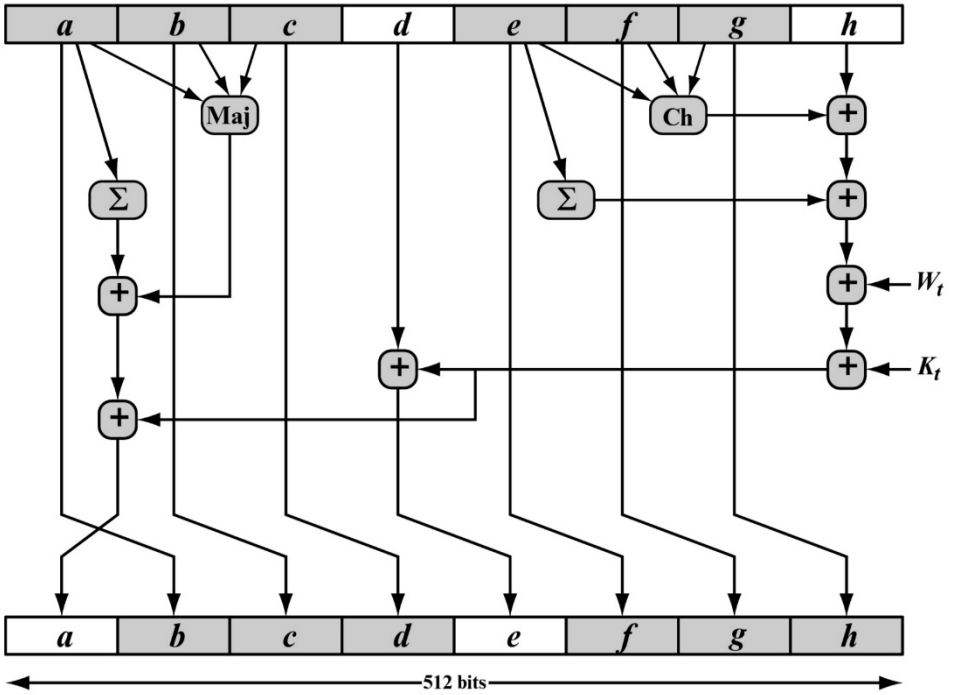
3.2.2 SHA-512

F:



Ch(e, f, g) = (e ∧ f) ⊕ (¬e ∧ g)

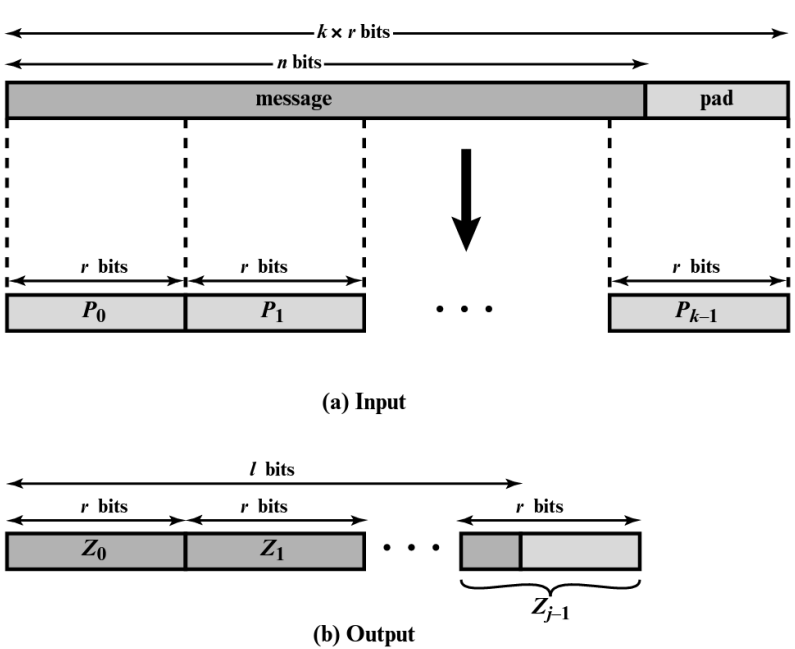
Maj(a, b, c) = (a ∧ b) ⊕ (a ∧ c) ⊕ (b ∧ c)



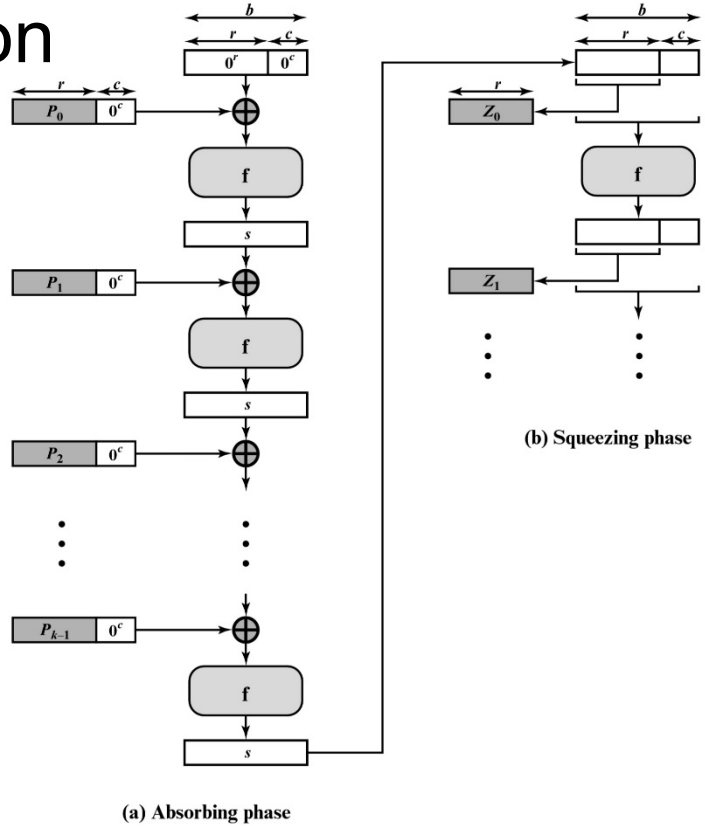
UNIVERSITÄT BERN

2. Hashing

3.3.1 SHA-3: Sponge Construction

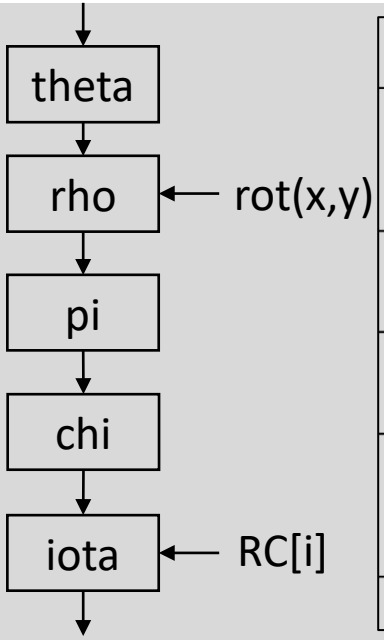


state variable s



2. Hashing

3.3.2 SHA-3: Iteration Function f

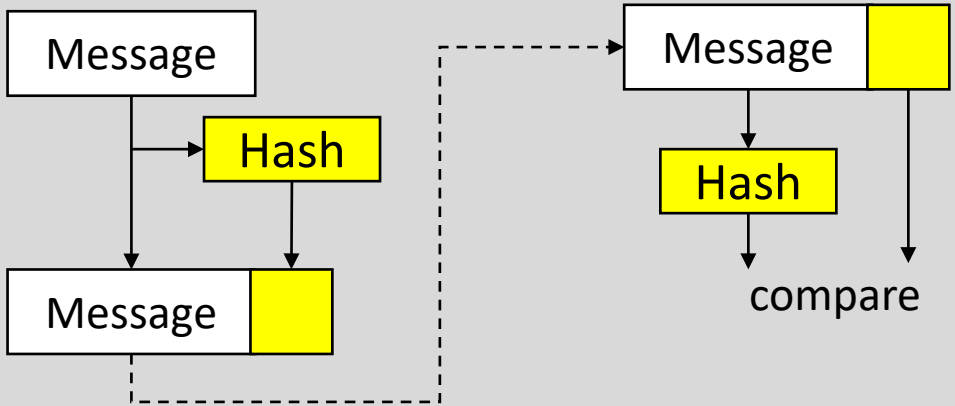


Function	Type	Description
θ	Substitution	New value of each bit in each word depends on its current value and on one bit in each word of preceding column and one bit of each word in succeeding column.
ρ	Permutation	The bits of each word are permuted using a circular bit shift. $W[0, 0]$ is not affected.
π	Permutation	Words are permuted in the 5×5 matrix. $W[0, 0]$ is not affected.
χ	Substitution	New value of each bit in each word depends on its current value and on one bit in next word in the same row and one bit in the second next word in the same row.
ι	Substitution	$W[0, 0]$ is updated by XOR with a round constant.

RC: round constant

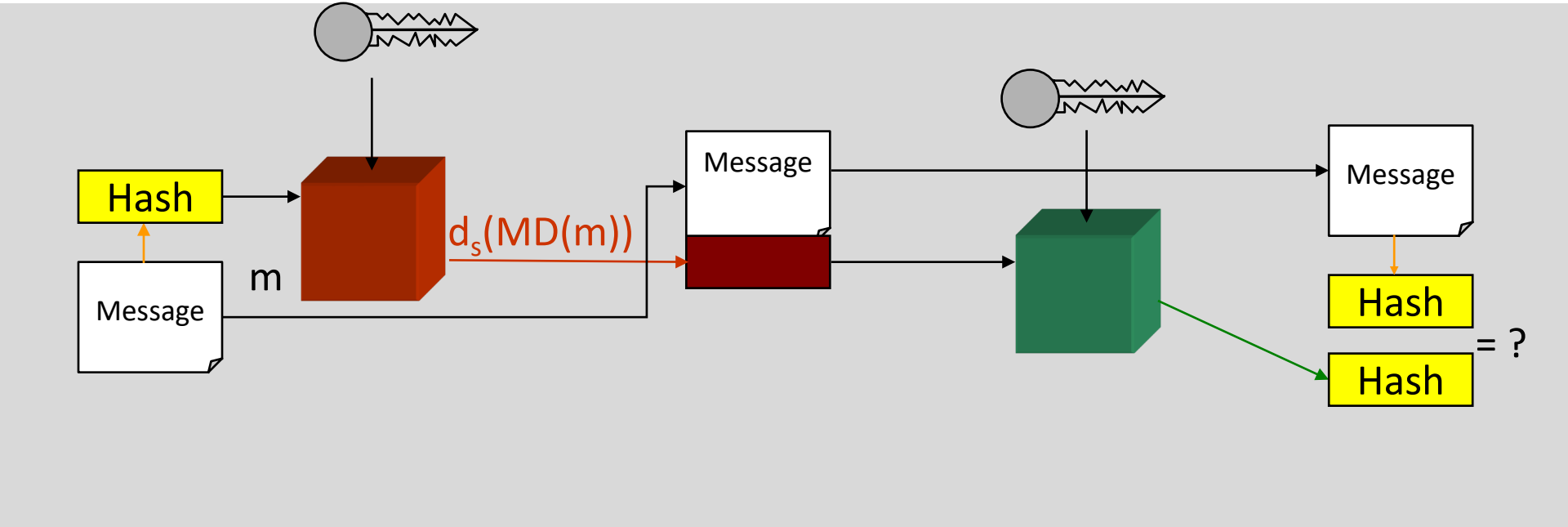
2. Hashing

4.1 Applications: Message Authentication Code



2. Hashing

4.2 Applications: Digital Signatures



2. Hashing

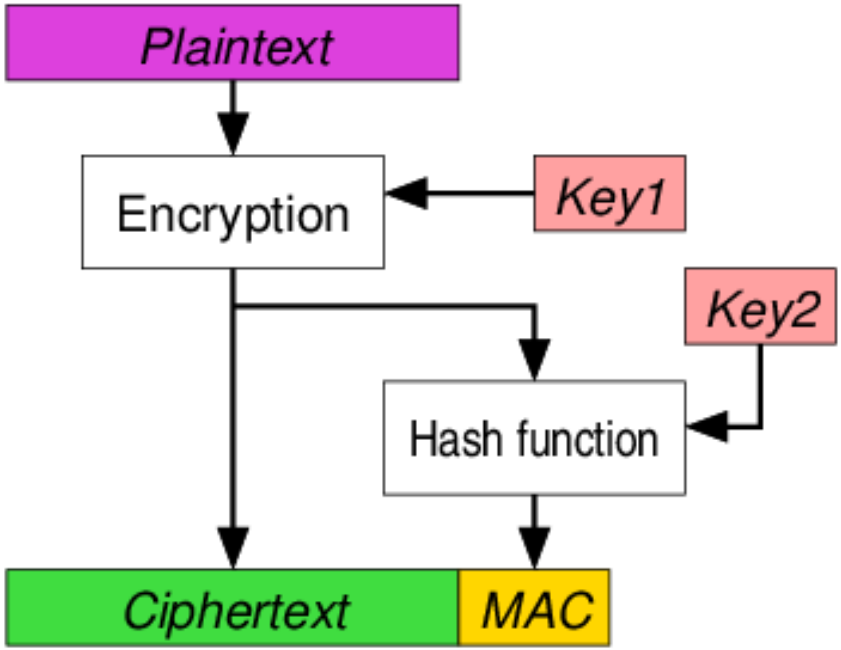
4.3 Applications: Authenticated Encryption

1. Encrypt-then-MAC
2. Encrypt-and-MAC
3. MAC-then-Encrypt

2. Hashing

4.3.1 Encrypt-then-MAC

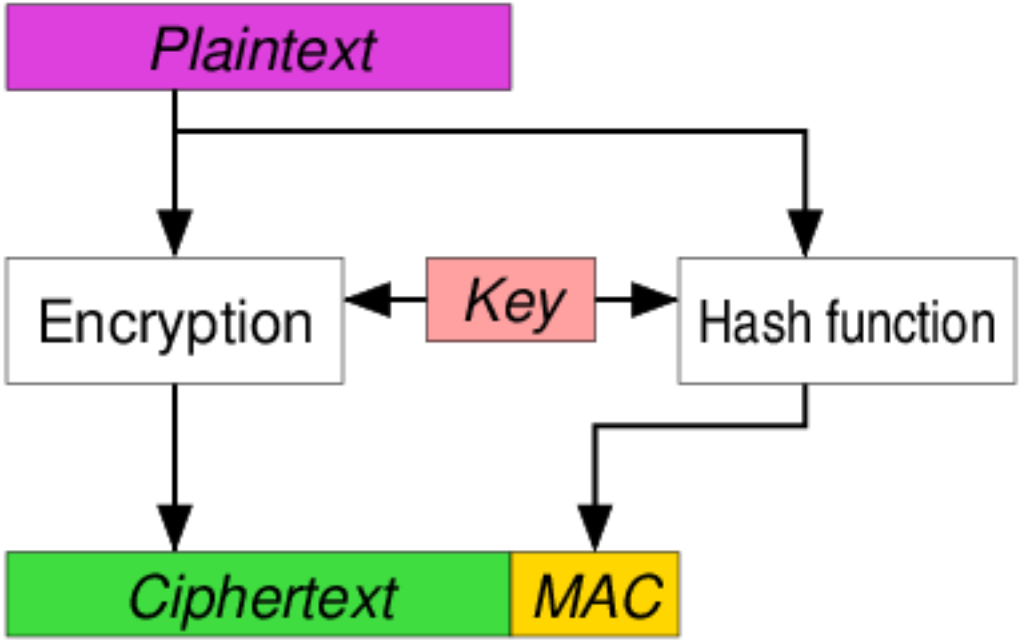
- Only method that can be proved to achieve the highest level of security
- used in TLS, SSH



2. Hashing

4.3.2 Encrypt-and-MAC

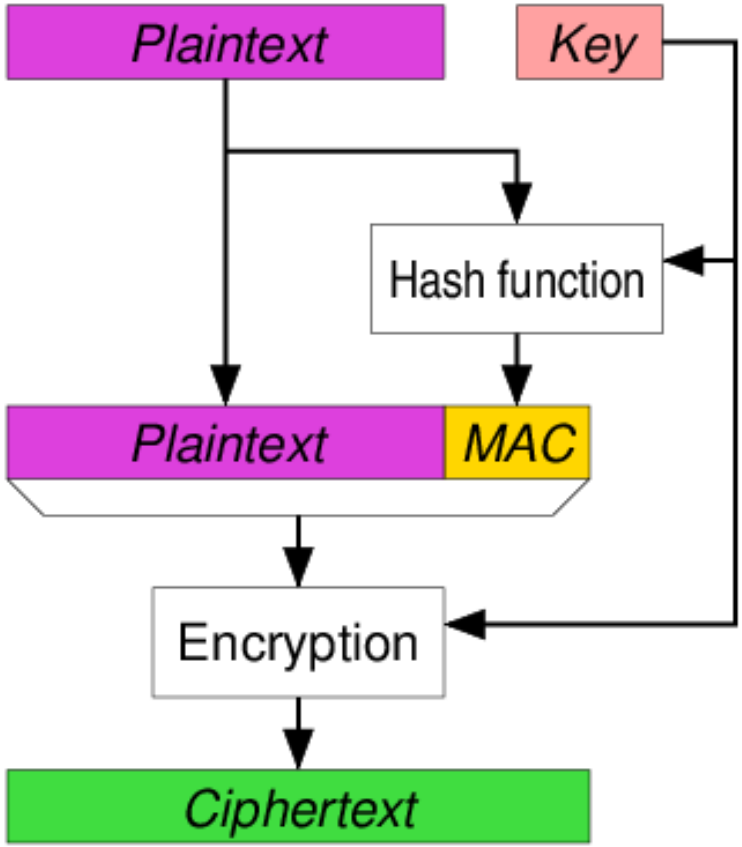
- can be made strongly secure
- used in SSH



2. Hashing

4.3.3 MAC-then-Encrypt

- can be made strongly secure
- used in TLS

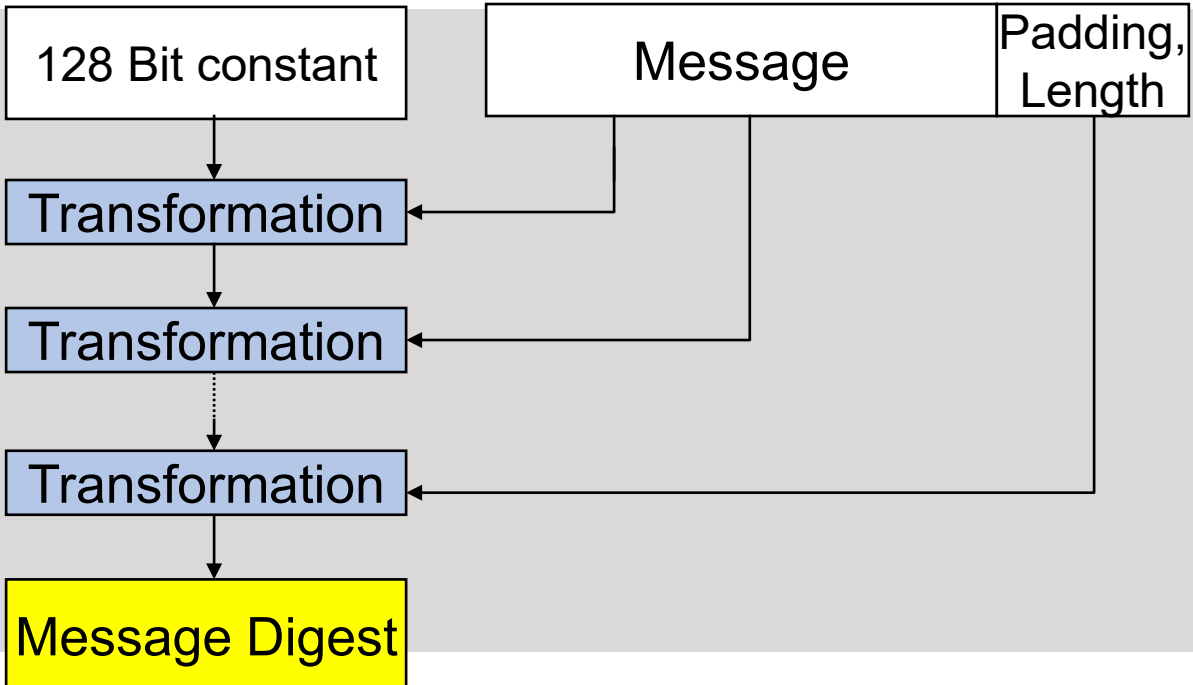


2. Hashing

4.4.1 Applications: Message Digest 5

Steps

1. Padding to length 448 mod 512
2. Append 64 bit length information
3. Initialization of 4 word buffer
   - A: 01 23 45 67
   - B: 89 ab cd ef
   - C: fe dc ba 98
   - D: 76 54 32 10
4. Processing in 512 bit steps



UNIVERSITÄT BERN

2. Hashing

4.4.2 Applications: MD5 Implementation

- Word register initializations
  - A: 01 23 45 67; B: 89 ab cd ef;
    C: fe dc ba 98; D: 76 54 32 10
- Notation: [abcd k s i]
- Variable X
- Table T constructed from sine function
- Auxiliary functions
  - $F(X,Y,Z) = XY \vee (\neg X \wedge Z)$
  - $G(X,Y,Z) = XZ \vee (Y \wedge \neg Z)$
  - $H(X,Y,Z) = X \oplus Y \oplus Z$
  - $I(X,Y,Z) = Y \oplus (X \vee \neg Z)$

```
/* Round 1; a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s) */
[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]

/* Round 2; a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s) */
[ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]
[ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]
[ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]
[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]

/* Round 3; a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s) */
[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]
[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]

/* Round 4; a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s) */
[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]
[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]
[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]
```

Thanks

for Your Attention

Prof. Dr. Torsten Braun, Institut für Informatik

Bern, 14.03.2022 – 21.03.2022