Using the AES round beyond encryption

Léo Perrin[1]

[1]Inria, France

1st AES retreat, April 2026

What can the efficiency of AES-NI and the ease of analysis of the AES give outside of crytograhy?

What can the efficiency of AES-NI and the ease of analysis of the AES give outside of crytograhy?

(or, equivalently, let's get rid of SipHash and the Mersenne Twister)

Outline

Plan of this Section

Versatility and Ease of Analysis

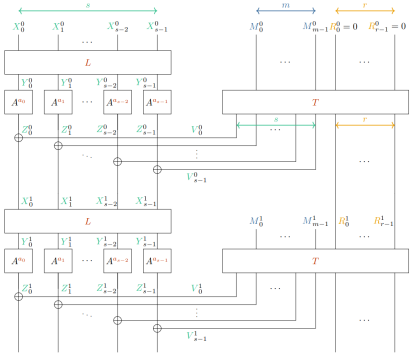


Figure 1: A stands for a key-less AES round. Each choice of the size parameters s, m, r, the Boolean values a_i, and the linear matrices L, T defines an instance of the framework.

The probability of all differential trails can be bounded in reasonable time for all such round functions!

Bariant, A., Baudrin, J., Leurent, G., Pernot, C., Perrin, L., & Peyrin, T. (2024). Fast AES-Based Universal Hash Functions and MACs: Featuring LeMac and PetitMac. IACR Transactions on Symmetric Cryptology, 2024(2), 35-67.

Versatility and Ease of Analysis

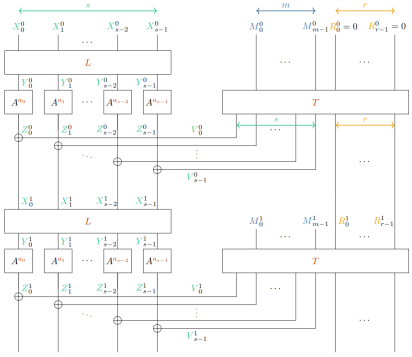


Figure 1: A stands for a key-less AES round. Each choice of the size parameters s, m, r, the Boolean values a_i, and the linear matrices L, T defines an instance of the framework.

The probability of all differential trails can be bounded in reasonable time for all such round functions!

$\rightarrow$ we can automatically lower bound # active S-boxes

Bariant, A., Baudrin, J., Leurent, G., Pernot, C., Perrin, L., & Peyrin, T. (2024). Fast AES-Based Universal Hash Functions and MACs: Featuring LeMac and PetitMac. IACR Transactions on Symmetric Cryptology, 2024(2), 35-67.

AES-NI: speed!!!

Table 2: Table of the retained candidates over different parameters sets. Speeds were measured on a Intel 11th Gen Core i5-1135G7 (Tiger Lake) for different sizes of message.

								Speed (cy/B)		
Rate	w	m	s	r	XOR-cost	Diffusion	Security	16 kB	256 kB	Descr.
2	8	4	9	4	4	15	26	0.074	0.067	Figure 2
1.75	7	4	10	5	5	∞	23	0.079	0.068	App. C.1
2	6	3	7	4	4	11	25	0.086	0.080	App. C.2
2	4	2	6	4	3	9	24	0.104	0.099	App. C.3
2	2	1	4	3	4	5	23	0.180	0.175	App. C.4
2	1	0.5[1]	1	5	3/1[2]	-	26	0.374	0.371	Figure 3

[1] A message is added every other round.

[2] There is 1 inherent XOR in the transition matrix. Every other rounds, the message accounts for 2 additional XORS.

Plan of this Section

Outside Cryptography

1. Monte Carlo simulations
2. Data set for proper software tests (fuzzing)
3. ...

In Cryptography too!

Entropy Distribution Function

First introduced by Saarinen[1]

Properties. The main security requirement for a lightweight mixing function is captured in the term "Entropy Distribution Function" (EDF); once seeded with $n \leq c$ truly random bits (n bits of entropy), any n-bit output should also have close to n bits of randomness (entropy) when observed without joint information.

SNEIGEN is **not** claimed to be collision resistant, but full collisions are unlikely for outputs that are much larger than the c-bit input seed. Given more than c bits of output, an attacker may be able to distinguish SNEIGEN from random, and may also be able to derive the secret state or even the input seed from it. However, targeted cryptanalytic effort is required to achieve this. SNEIGEN output should not be directly exposed to an attacker in a way that leads to the compromise of secret state.

Since the SNEIK permutation has a $2 \times 32 = 64$ - bit feedback "accumulator" (words $\mathsf{s}[i-1]$ and $\mathsf{s}[i-2]$ in Equation 2) diffusion to the right, and the first round does not achieve much diffusion to the left, the number of rounds is chosen as $\rho = \lfloor c/64 \rfloor + 1$.

[1] M. J. Saarinen, *SNEIKEN and SNEIKHA*, submission to NIST LWC competition, 2019

`https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/sneik-spec.pdf`

Mersenne Twister is everywhere

```
uint32_t random_uint32(mt_state* state)
{
    uint32_t* state_array = &(state->state_array[0]);

    int k = state->state_index;       // point to current state location
                                      // 0 <= state_index <= n-1   always

//  int k = k - n;                    // point to state n iterations before
//  if (k < 0) k += n;                // modulo n circular indexing
                                      // the previous 2 lines actually do nothing
                                      //  for illustration only

    int j = k - (n-1);                // point to state n-1 iterations before
    if (j < 0) j += n;                // modulo n circular indexing

    uint32_t x = (state_array[k] & UMASK) | (state_array[j] & LMASK);

    uint32_t xA = x >> 1;
    if (x & 0x00000001UL) xA ^= a;

    j = k - (n-m);                    // point to state n-m iterations before
    if (j < 0) j += n;                // modulo n circular indexing

    x = state_array[j] ^ xA;          // compute next value in the state
    state_array[k++] = x;             // update new state value

    if (k >= n) k = 0;                // modulo n circular indexing
    state->state_index = k;

    uint32_t y = x ^ (x >> u);        // tempering
             y = y ^ ((y << s) & b);
             y = y ^ ((y << t) & c);
    uint32_t z = y ^ (y >> l);

    return z;
}
```

Basically a glorified LFSR

- C++ stdlib
- python random
- ...

$O\left(\text{a dozen}\right)$ instructions/32 bits

Towards a Design?

- AES in counter mode?

Towards a Design?

- AES in counter mode?
- round-reduced AES in counter mode?

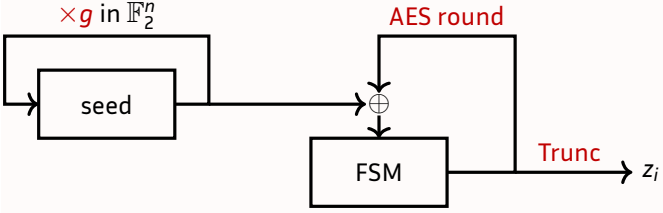
Towards a Design?

- AES in counter mode?
- round-reduced AES in counter mode?
- SNOW with a simplified FSM?

Towards a Design?

- AES in counter mode?
- round-reduced AES in counter mode?
- SNOW with a simplified FSM?

Something like this?



MILP-based linear trail analysis to lower bound correlations?

$O(3)$ instructions/64 bits

Plan of this Section

Hash functions for Hash Tables

Siphash

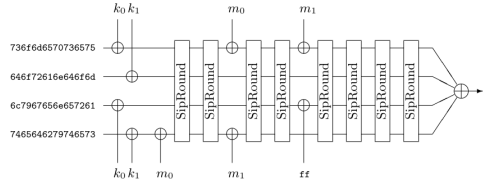


Fig. 2.1. SipHash-2-4 processing a 15-byte message. SipHash-2-4(k, m) is the output from the final $\oplus$ on the right.

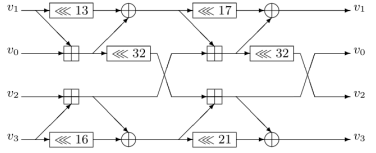


Fig. 2.2. The ARX network of SipRound.

Hash functions for Hash Tables

Siphash

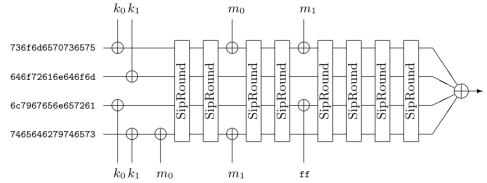


Fig. 2.1. SipHash-2-4 processing a 15-byte message. SipHash-2-4(k, m) is the output from the final $\oplus$ on the right.

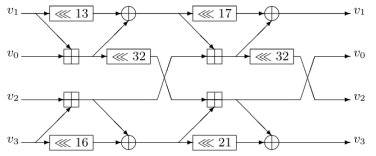


Fig. 2.2. The ARX network of SipRound.

Abseil Hash function

```
// We use each value as the first argument to shuffle all the bits around. We do
// not add any salt to the state or loaded data, instead we vary instructions
// used to mix bits Encrypt128/Decrypt128 and Add128/Sub128. On x86,
// Add128/Sub128 are combined to one instruction with data loading like
// `vpaddq xmm1, xmm0, xmmword ptr [rdi]`.

inline Vector128 MixA(Vector128 a, Vector128 state) {
  return Decrypt128(Add128(state, a), state);
}

inline Vector128 MixB(Vector128 b, Vector128 state) {
  return Decrypt128(Sub128(state, b), state);
}

inline Vector128 MixC(Vector128 c, Vector128 state) {
  return Encrypt128(Add128(state, c), state);
}

inline Vector128 MixD(Vector128 d, Vector128 state) {
  return Encrypt128(Sub128(state, d), state);
}
```

https://github.com/abseil/abseil-cpp/blob/master/absl/hash/internal/hash.cc

Towards a Design?

- PetitMAC with little/no initialization/finalization?

Towards a Design?

- PetitMAC with little/no initialization/finalization?

Something like this?

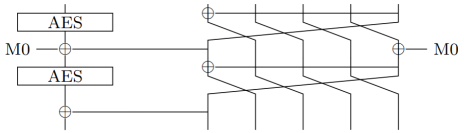


Figure 3: Processing one message block in the UHF used in PetitMac.

2 AES + 3 XOR / 128-bit block

Towards a Design?

- PetitMAC with little/no initialization/finalization?

Something like this?

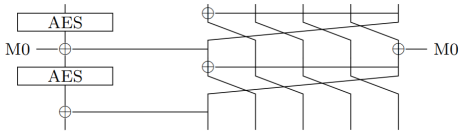


Figure 3: Processing one message block in the UHF used in PetitMac.

2 AES + 3 XOR / 128-bit block

lower bound on differential probabilities $\longrightarrow$ collision security

Plan of this Section

Conclusion

Thanks to AES-NI and decades of cryptanalysis, we could...

Conclusion

Thanks to AES-NI and decades of cryptanalysis, we could...

- do insecure algorithms :D

Conclusion

Thanks to AES-NI and decades of cryptanalysis, we could...

- do insecure algorithms :D
- do so much better than Siphash and the Mersenne twister!

Conclusion

Thanks to AES-NI and decades of cryptanalysis, we could...

- do insecure algorithms :D
- do so much better than Siphash and the Mersenne twister!

Thank You!