Hitag 2 Hell – Brutally Optimizing Guess-and-Determine Attacks

Aram Verstegen and Roel Verdult and Wouter Bokslag

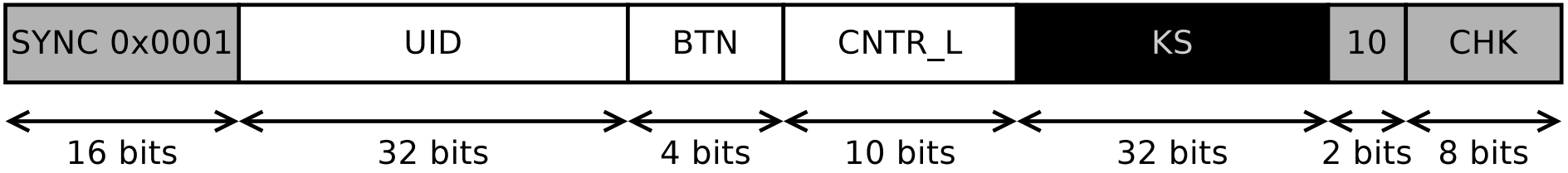
FactorIT B.V.

August 14th 2018

Overview

1. Introduction
2. LFSR-based stream ciphers
3. Hitag2
4. Guess-and-determine attacks
5. Performance optimization
6. Performance evaluation
7. Takeaways
8. End-to-end demonstration

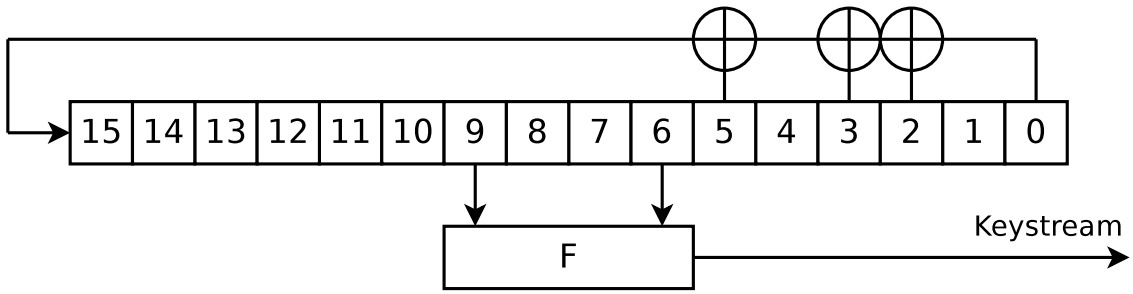
Remote Keyless Entry



UHF frame format

- Contains 32-bit MAC (`KS`) on frame data
- `KS` is a 32-bit sample of Hitag2 keystream
- Determined by the public `UID`, secret 48-bit key and partly public 32-bit nonce
- 28-bit counter is incremented after each press

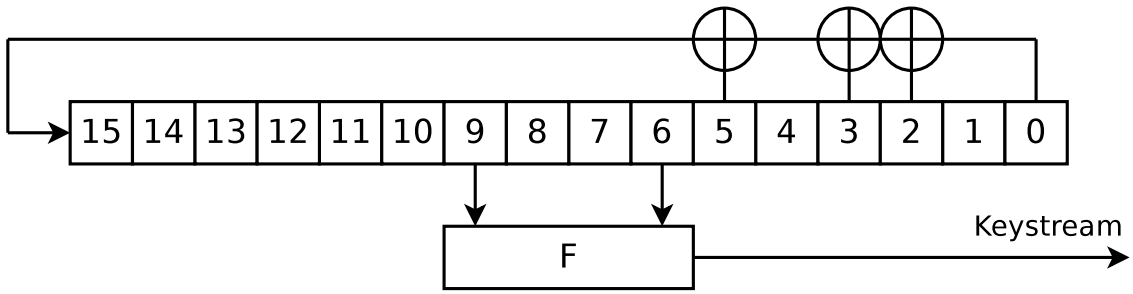
LFSR-based stream ciphers



LFSR

- State register is initialized with secret key
- Round function shifts register by one bit
- *Feedback function* computes new bit to shift in, based on state

LFSR-based stream ciphers



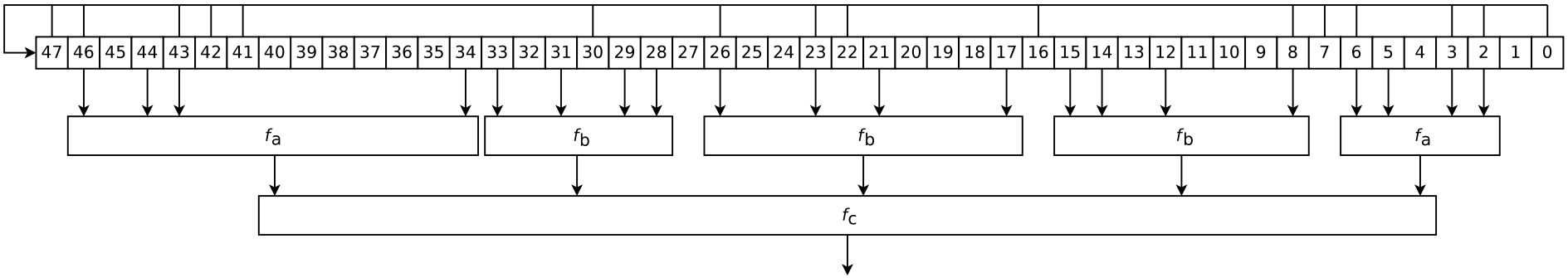
LFSR

- State register is initialized with secret key
- Round function shifts register by one bit
- *Feedback function* computes new bit to shift in, based on state

Cryptanalysis

- Try to find state at any point in time
- Construct inverse round function, roll back to initial state

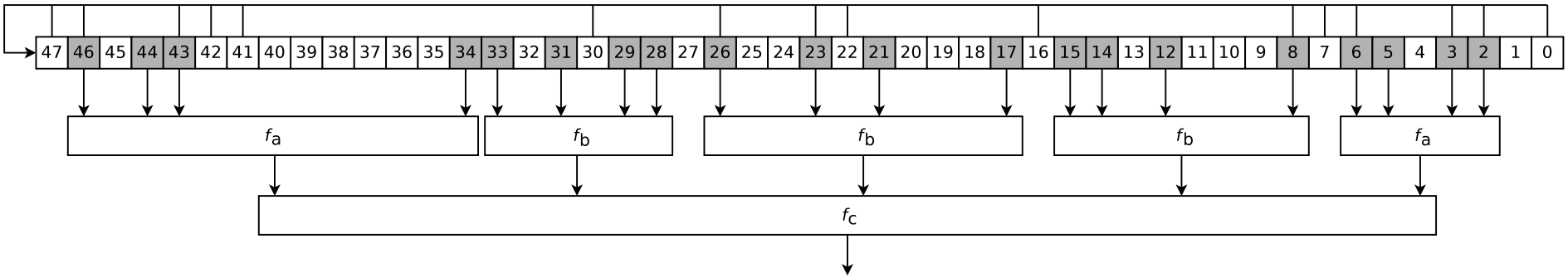
Hitag2



Hitag2 filter function

- Keystream generated by nonlinear filter function
- Consists of two layers of boolean operations
- Keystream bit value is determined by 20 state bits

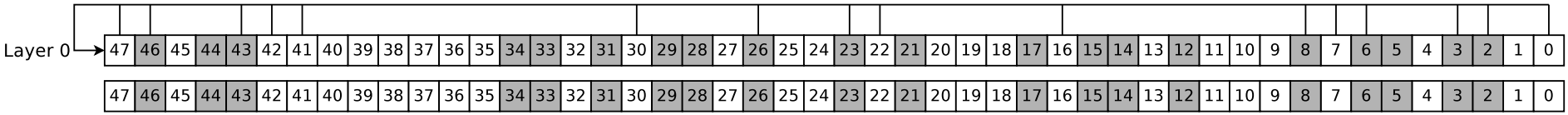
Hitag2



Leakage of state information

- Keystream bit reveals information about cipher state bits
- 2^{20} candidates, but only 2^{19} generate observed bit
- Eliminate half the possibilities for each keystream bit

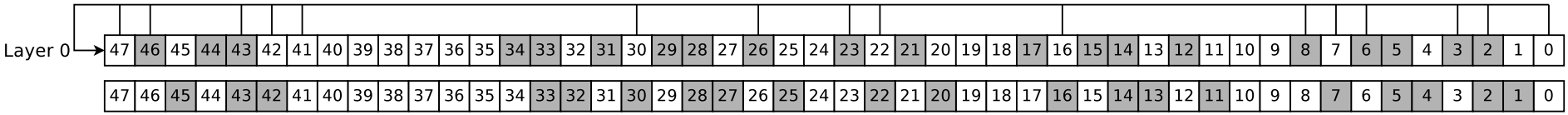
Guess-and-determine



Propagate guesses over multiple *layers*

- Several of the input bits for f can overlap with previous guesses
- Fewer guesses to make on the state for each subsequent keystream bit tested
- Still eliminate half the guesses in each layer
- Layer guesses can be expressed as bit masks

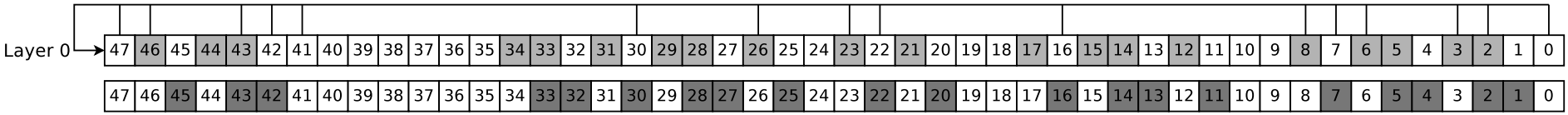
Guess-and-determine



Propagate guesses over multiple *layers*

- Several of the input bits for f can overlap with previous guesses
- Fewer guesses to make on the state for each subsequent keystream bit tested
- Still eliminate half the guesses in each layer
- Layer guesses can be expressed as bit masks

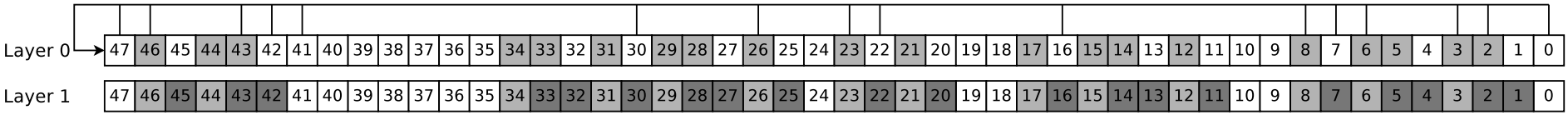
Guess-and-determine



Propagate guesses over multiple *layers*

- Several of the input bits for f can overlap with previous guesses
- Fewer guesses to make on the state for each subsequent keystream bit tested
- Still eliminate half the guesses in each layer
- Layer guesses can be expressed as bit masks

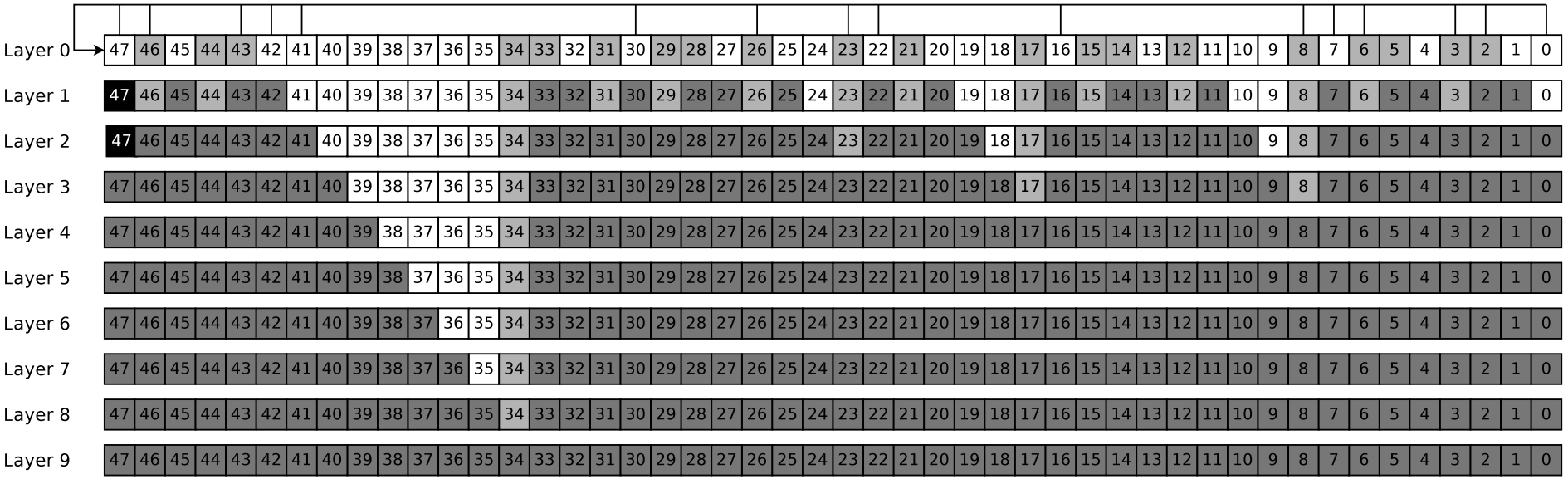
Guess-and-determine



Propagate guesses over multiple *layers*

- Several of the input bits for f can overlap with previous guesses
- Fewer guesses to make on the state for each subsequent keystream bit tested
- Still eliminate half the guesses in each layer
- Layer guesses can be expressed as bit masks

Guess-and-determine



Full state is determined at layer 9

- 2^{39} states reach this point on average
- Test 23 more keystream bits to reduce to $\sim 2^{16}$
- Internal states can be rolled back to initial state with inverse LFSR

Performance optimization

Trivial optimizations

- Pre-compute filter function outputs through lookup tables
- Pre-compute state bit guesses
- Unroll 9 layers of recursion as 9 nested loops

Performance optimization

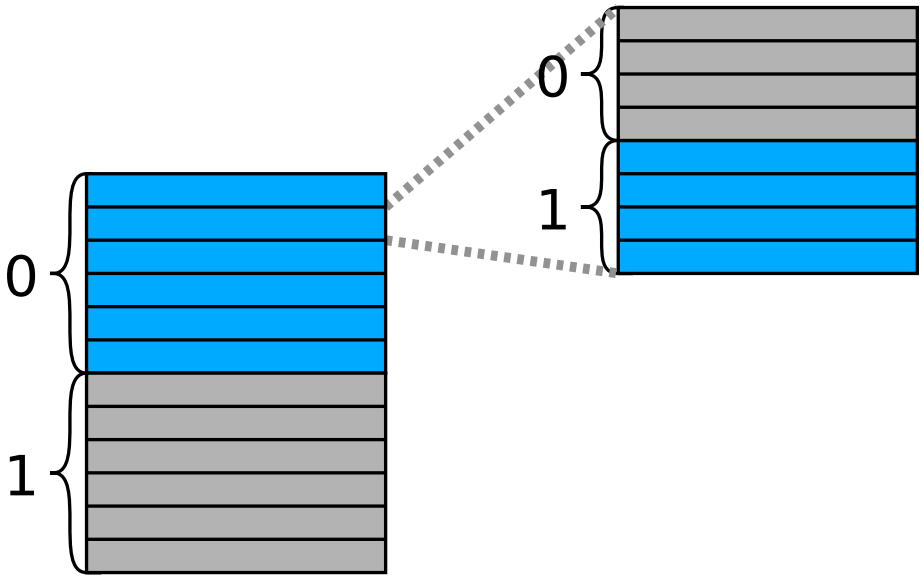
Trivial optimizations

- Pre-compute filter function outputs through lookup tables
- Pre-compute state bit guesses
- Unroll 9 layers of recursion as 9 nested loops

Advanced optimizations

- Only generate ideal guesses
- Pre-compute and cache the results for later computations in earlier ones
- Bit-slice the implementation

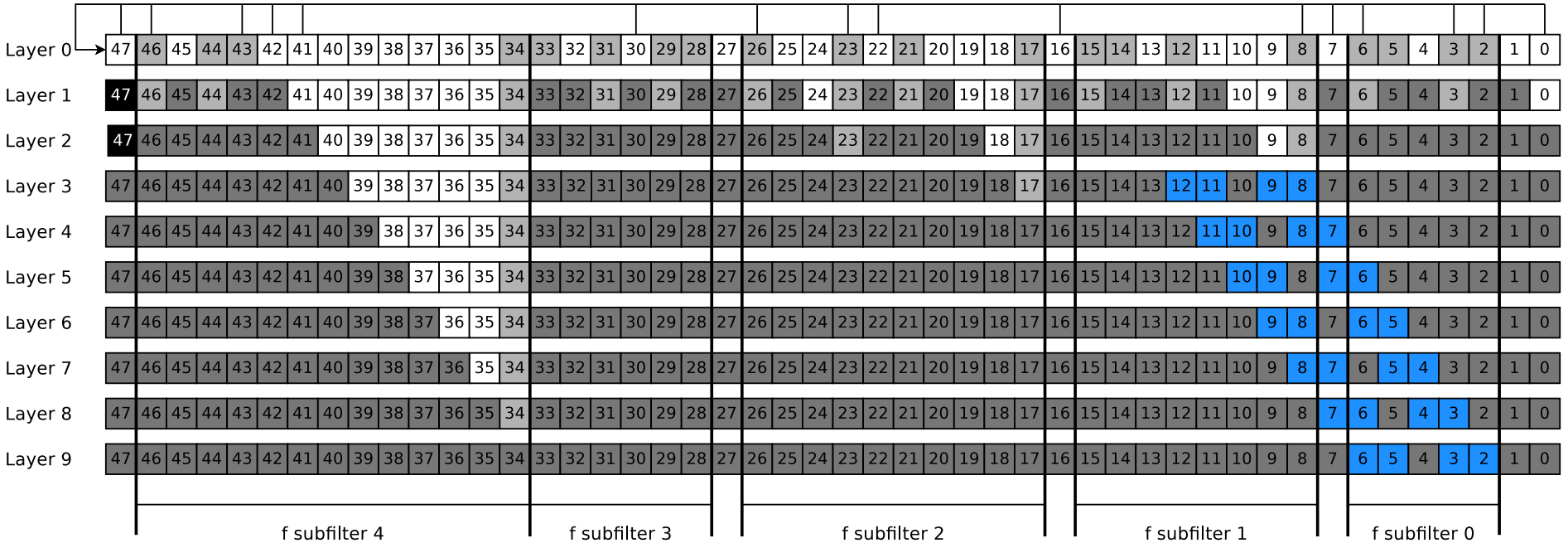
Performance optimization



Generating ideal guesses

- Use known state information and expected result to generate ideal lookup tables per layer
- Can test first 9 keystream bits without relying on f entirely
- 23 MB tables too large to fit in cache – negative trade-off

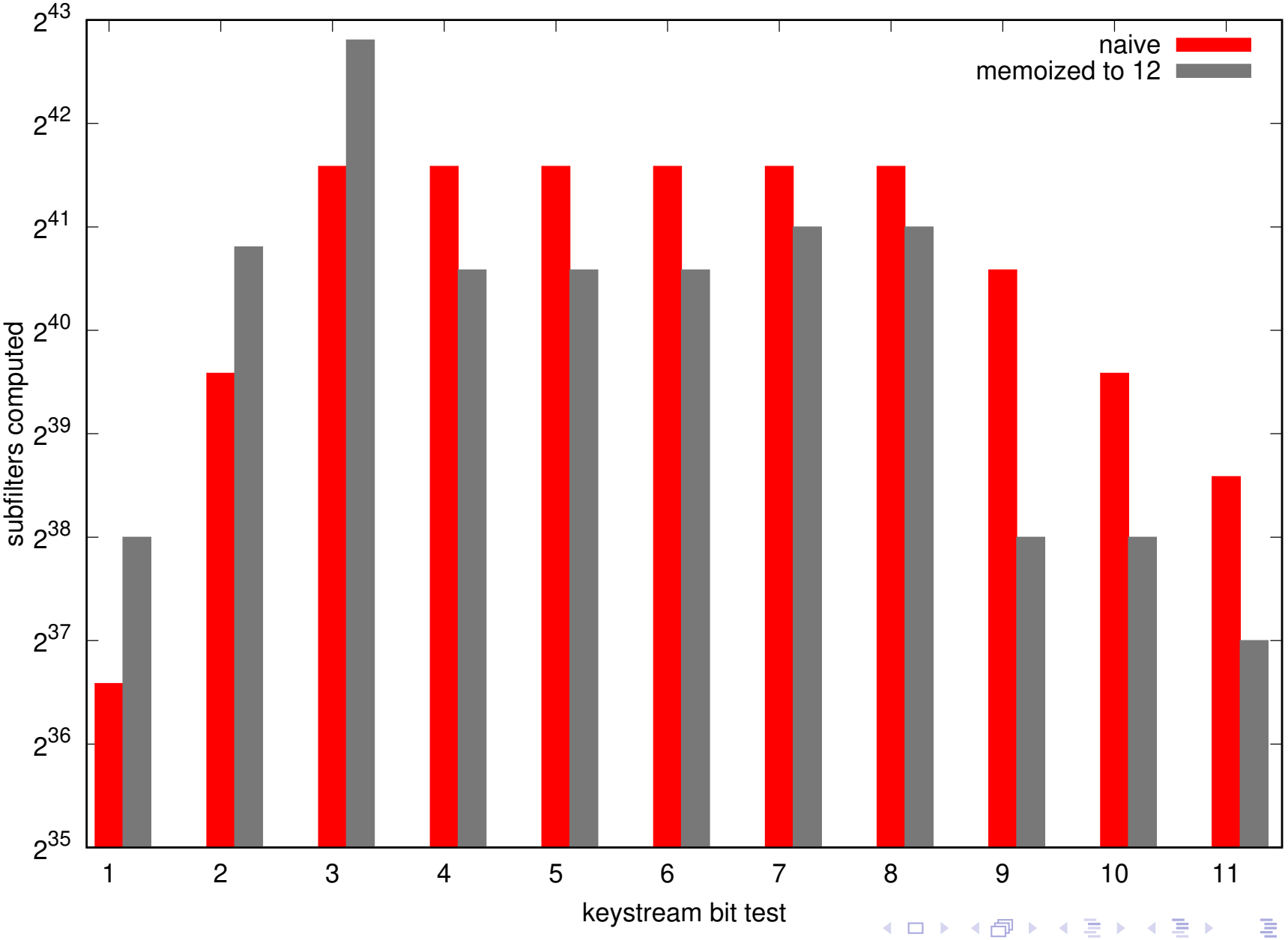
Performance optimization



Memoization

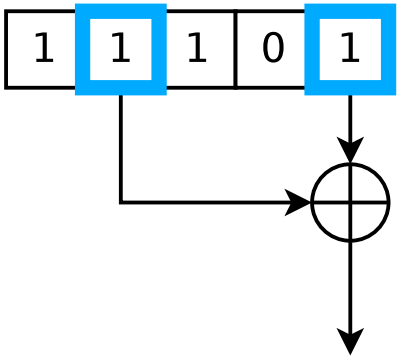
- From layer 2 onwards, some subfilters for upcoming layers are entirely determined
- Keep *memos* of the evaluation of these subfilters (*memoization*)

Performance optimization



Performance optimization

LFSR view

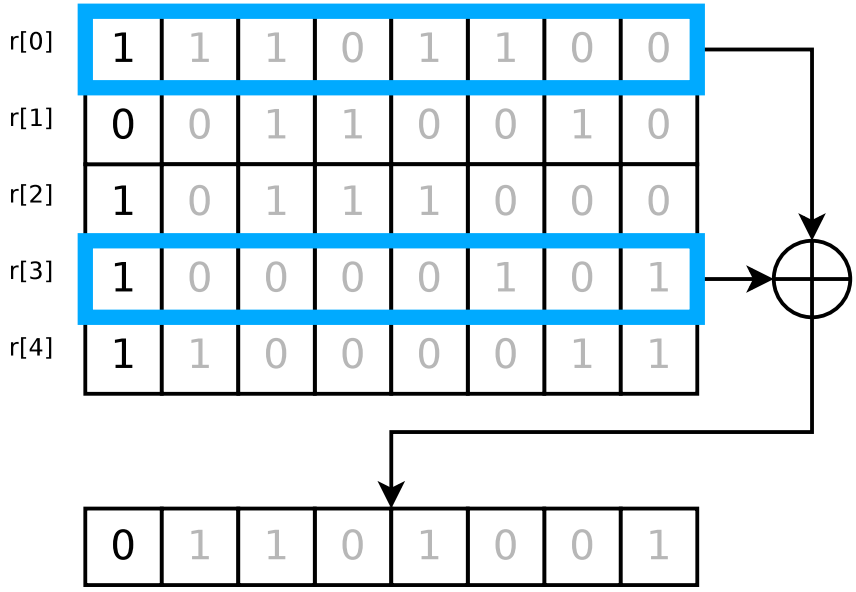


Many LFSRs are evaluated

r[0]	1	1	1	0	1
r[1]	1	0	0	0	1
r[2]	0	0	1	1	1
r[3]	0	0	1	1	0
r[4]	0	0	1	0	1
r[5]	0	1	0	0	1
r[6]	1	0	0	1	0
r[7]	1	1	0	0	0

```
for (i = 0; i < 8, i++)
    ans[i] = ((r[i] & 0b1000) >> 3) ^ r[i] & 1
```

Transpose bits in registers



ans $_{0\ ..\ 7}$= r[0] ^ r[3]

Bit-slicing

- Parallelize to native register width
- Transpose multiple states over multiple v-bit registers

Performance evaluation

Optimization type	Platform	Cores	Running time
Naive BF (Immler 2012)	Tesla C2050	448 CUDA	660 minutes
32-way bit-sliced (Benadjila et al. 2017)	GTX780Ti	2880 CUDA	1080 minutes
32-way bit-sliced (Benadjila et al. 2017)	16 x Tesla K80	79872 CUDA	45 minutes
Naive guess-and-determine	i7 3700K	4 Intel	127 minutes
Avoiding impossible guesses	i7 3700K	4 Intel	96 minutes
256-way bit-sliced, memoized to depth 12	i7 3700K	4 Intel	59 minutes
32-way bit-sliced, memoized to depth 17	GTX1080Ti	3584 CUDA	**1.2 minutes**

Table: Performance evaluation of average running time compared to related work.

Performance

- Two previous publications implement naive brute force
- Guess-and-determine offers serious performance improvement
- Factor 500 performance improvement[a]

[a] Normalizing for cuda core count and gpu base clock speed

Takeaways

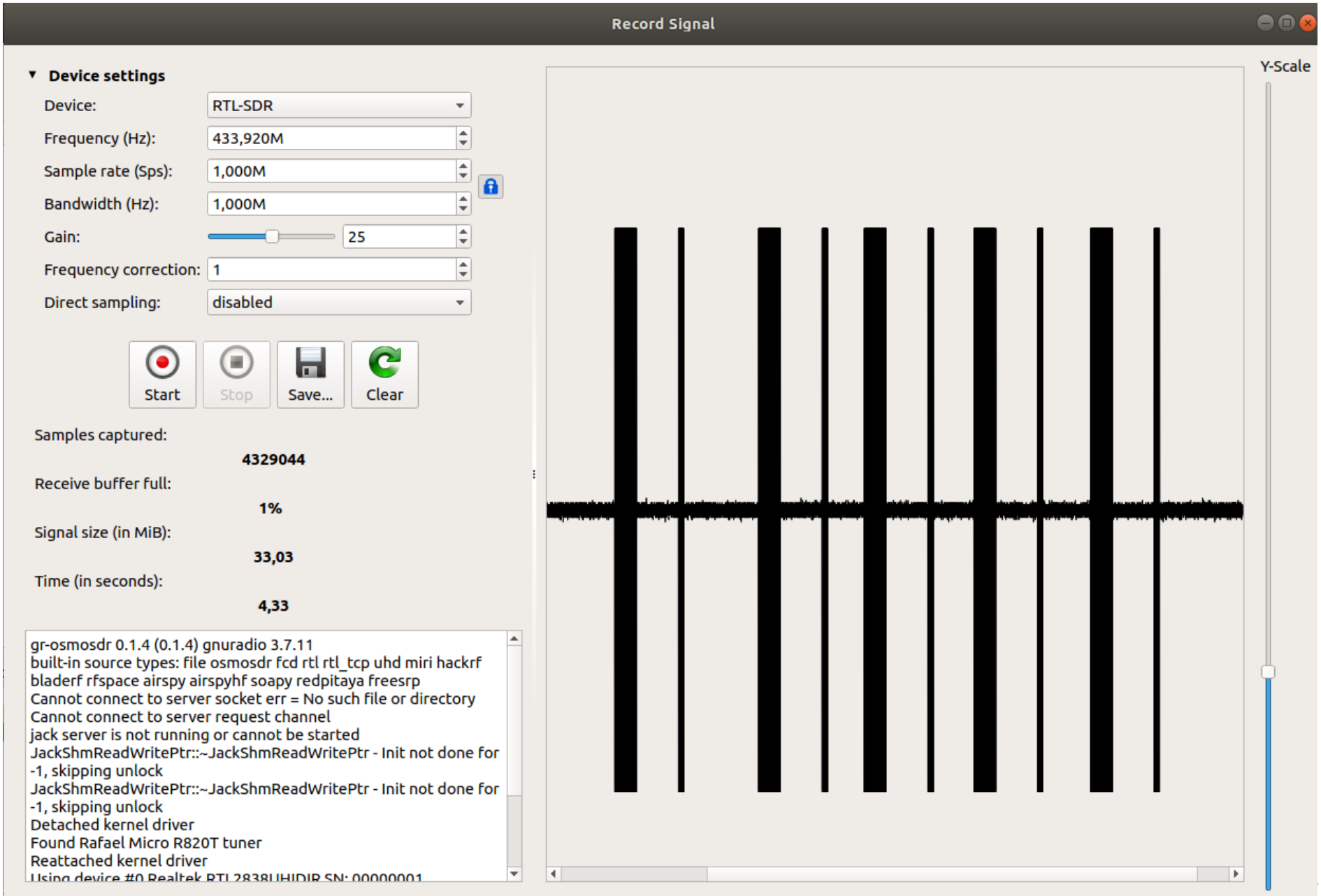
Guess-and-determine optimization techniques

- Conceptual (predetermine guesses, memoized)
- Implementational (unrolled recursion, lookups, bit-sliced)

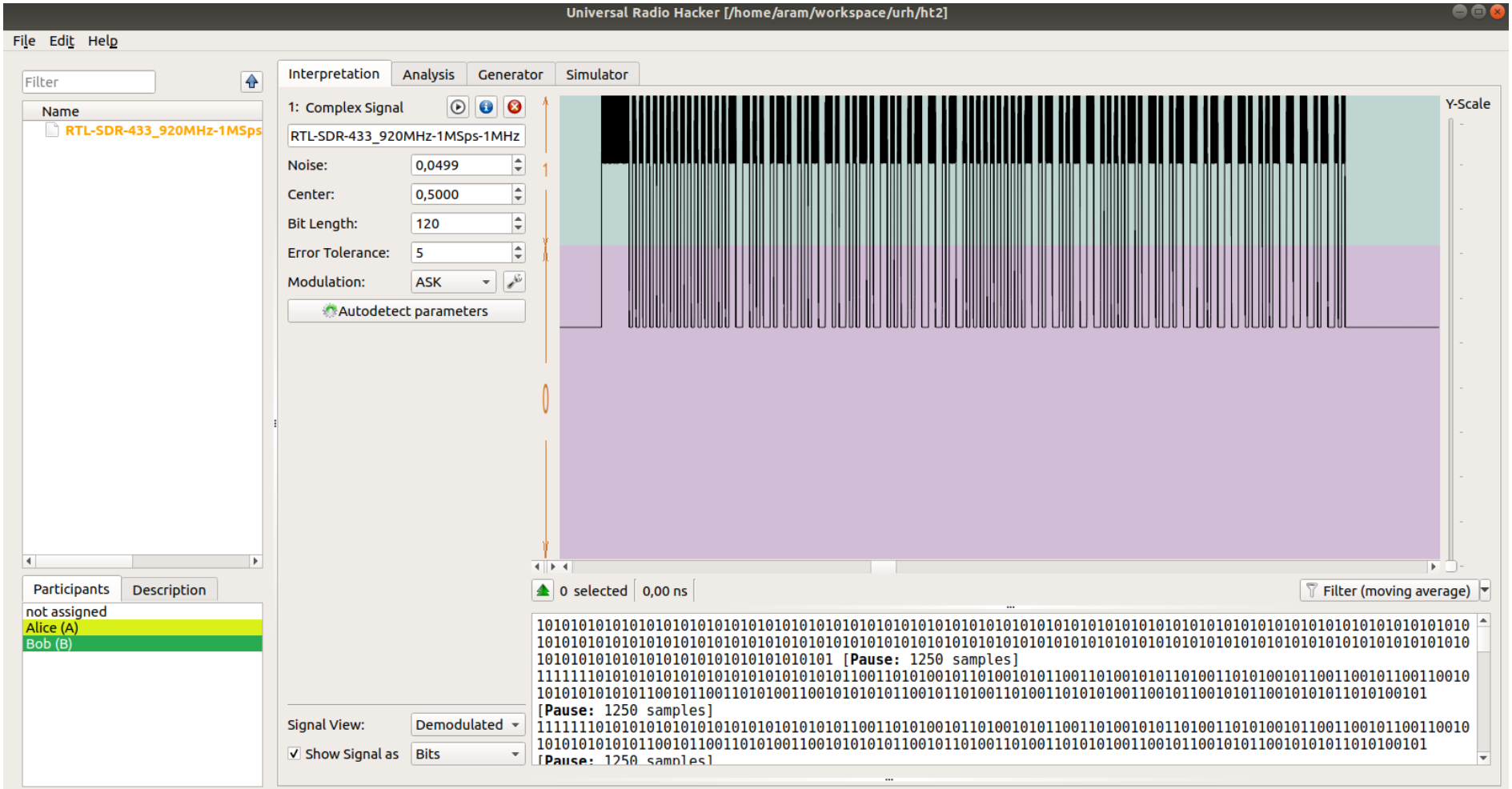
Proof of concept on Hitag2 stream cipher

- Recovers internal state in roughly 1 minute
- Applicable in RKE as well as in immobilizer context

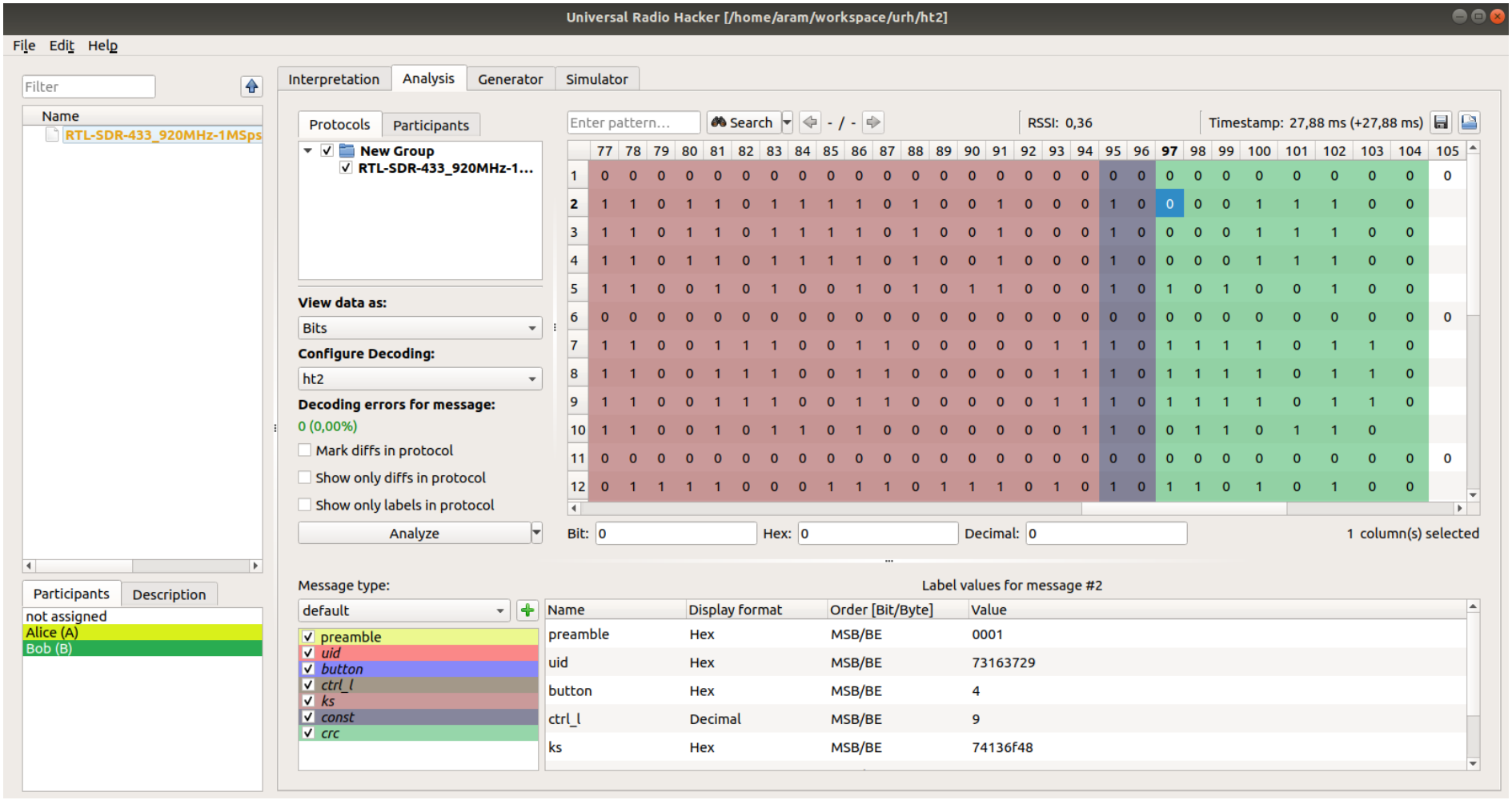
Demonstration



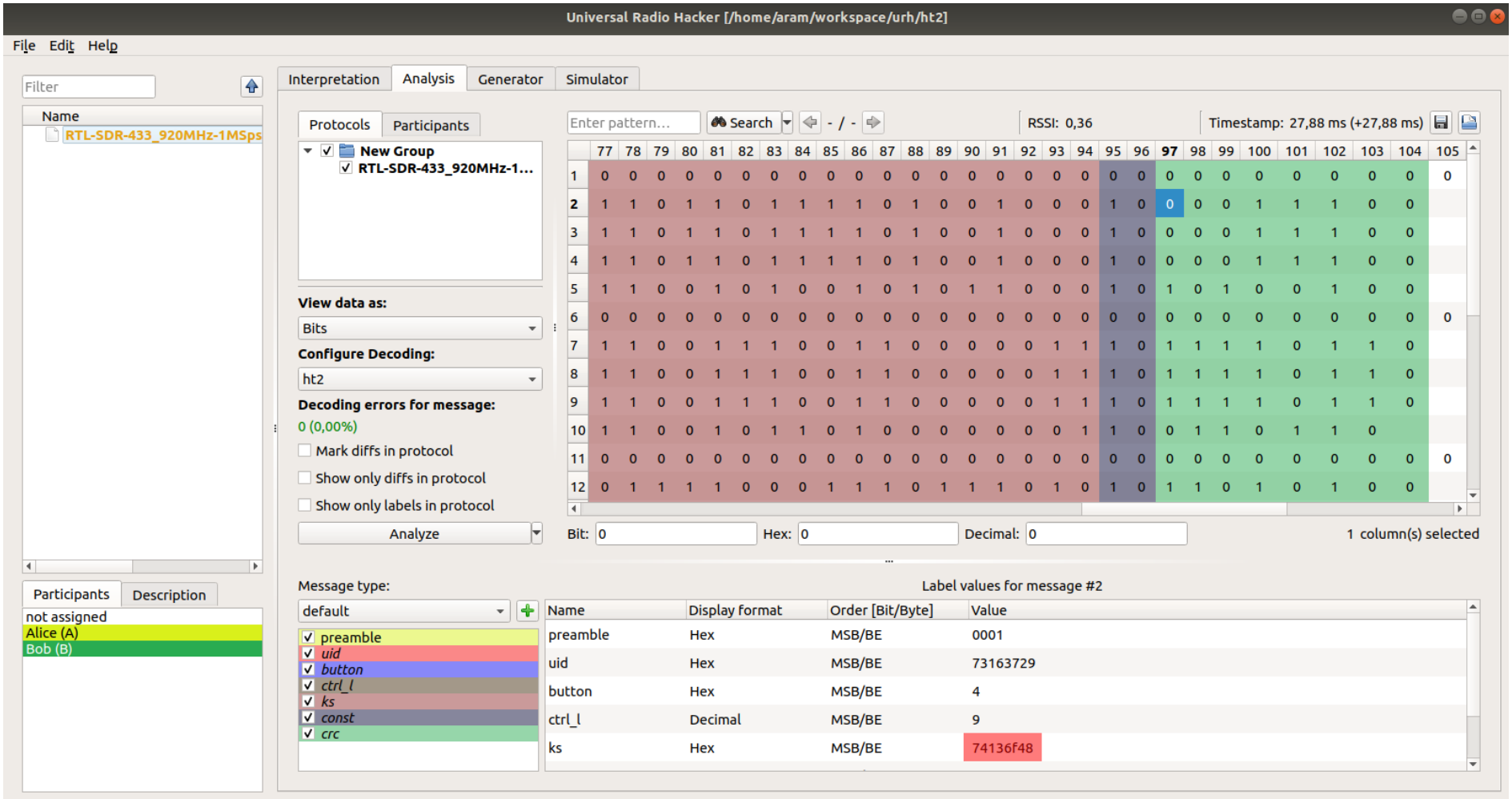
Demonstration



Demonstration



Demonstration



Keystream: `0x74136f48`

Demonstration

```
$ ./hitag2_cracker 74136f48
[+] FactorIT HITAG2 state finder
[i] Using platform: NVIDIA CUDA
[+] Cracker started
[i] Running Hitag2 on 1 device (25.00%)
[i] Running Hitag2 on 1 device (50.00%)
[i] Running Hitag2 on 1 device (75.00%)
[+] Cracker finished
Writing outfile 74136f48.txt

$ head 74136f48.txt
09a0245fcb36
0880245f0481
0f000ef0b3e3
...
```

Download

http://www.hitag2hell.com/