LUND
UNIVERSITY

Attacks on the Firekite Cipher

Thomas Johansson, Willi Meier, Vu Nguyen
Dept. of EIT, Lund University, FHNW

Motivation of the Firekite cipher

- Post-quantum cryptography is an active and relevant area.

Motivation of the Firekite cipher

- Post-quantum cryptography is an active and relevant area.
- Hard problems in post-quantum cryptography such as: code-based, lattice-based, multivariate-based...

Motivation of the Firekite cipher

- Post-quantum cryptography is an active and relevant area.
- Hard problems in post-quantum cryptography such as: code-based, lattice-based, multivariate-based...
  - Alternatively, learning assumptions such as LPN, LWE.

Motivation of the Firekite cipher

- Post-quantum cryptography is an active and relevant area.
- Hard problems in post-quantum cryptography such as: code-based, lattice-based, multivariate-based...
  - Alternatively, learning assumptions such as LPN, LWE.
- Learning Parity with noise is appealing in many applications for its simplicity.

Overview of LPN-based constructions

Common drawback:

- Often require fresh randomness (cryptographically secure bits) [Sho99, HDWH12].

Overview of LPN-based constructions

Common drawback:

- Often require fresh randomness (cryptographically secure bits) [Sho99, HDWH12].
- Not suitable for low-weight, restrained devices.

The Firekite cipher (Bogos et al.)

Design [BKLV21]

Assume $n \gg m$, and k are integers.

$$\overset{m}{\mathbf{v}} \cdot \overset{m \times n}{\mathbf{M}} + \underset{\uparrow \atop \text{error vector of weight } k}{\overset{n}{\mathbf{e}}} = \begin{pmatrix} \mathbf{g} \\ \mathbf{v}' \\ \mathbf{c}_e \end{pmatrix}$$

Figure: Overview of Firekite's design.

Design [BKLV21]

Assume $n \gg m$, and k are integers.

$$\overset{m}{\mathbf{v}} \cdot \overset{m \times n}{\mathbf{M}} + \underset{\text{error vector of weight } k}{\overset{n}{\mathbf{e}}} = \begin{pmatrix} \overset{\text{noisy product of length } n}{\mathbf{g}} \\ \mathbf{v}' \\ \mathbf{c}_e \end{pmatrix}$$

Figure: Overview of Firekite's design.

Design [BKLV21]

Assume $n \gg m$, and k are integers.

noisy product of length n

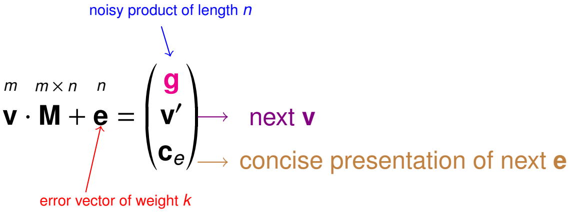
$$\overset{m}{\mathbf{v}} \cdot \overset{m \times n}{\mathbf{M}} + \overset{n}{\mathbf{e}} = \begin{pmatrix} \mathbf{g} \\ \mathbf{v}' \\ \mathbf{c}_e \end{pmatrix} \longrightarrow \text{next } \mathbf{v}$$

error vector of weight k

Figure: Overview of Firekite’s design.

Design [BKLV21]

Assume $n \gg m$, and k are integers.



noisy product of length n

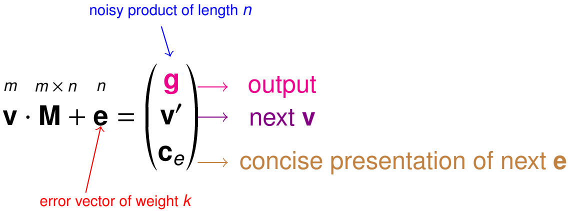
$$\overset{m}{\mathbf{v}} \cdot \overset{m \times n}{\mathbf{M}} + \overset{n}{\mathbf{e}} = \begin{pmatrix} \mathbf{g} \\ \mathbf{v}' \\ \mathbf{c}_e \end{pmatrix} \begin{matrix} \\ \longrightarrow \text{ next } \mathbf{v} \\ \longrightarrow \text{ concise presentation of next } \mathbf{e} \end{matrix}$$

error vector of weight k

Figure: Overview of Firekite’s design.

Design [BKLV21]

Assume $n \gg m$, and k are integers.



noisy product of length n

$$\overset{m}{\mathbf{v}} \cdot \overset{m \times n}{\mathbf{M}} + \underset{\text{error vector of weight } k}{\overset{n}{\mathbf{e}}} = \begin{pmatrix} \mathbf{g} \\ \mathbf{v}' \\ \mathbf{c}_e \end{pmatrix} \begin{matrix} \longrightarrow \text{ output} \\ \longrightarrow \text{ next } \mathbf{v} \\ \longrightarrow \text{ concise presentation of next } \mathbf{e} \end{matrix}$$

Figure: Overview of Firekite's design.

Design [BKLV21]

Assume $n \gg m$, and k are integers.

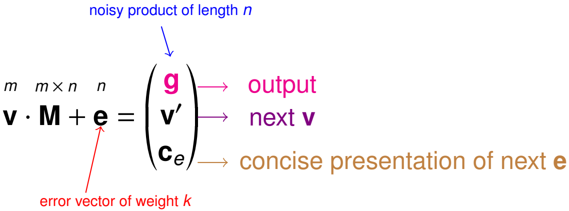
$$\overset{m}{\mathbf{v}} \cdot \overset{m\times n}{\mathbf{M}} + \underset{\text{error vector of weight } k}{\overset{n}{\mathbf{e}}} = \overset{\text{noisy product of length } n}{\begin{pmatrix}\mathbf{g}\\ \mathbf{v}'\\ \mathbf{c}_e\end{pmatrix}} \begin{matrix}\longrightarrow \text{output}\\ \longrightarrow \text{next } \mathbf{v}\\ \longrightarrow \text{concise presentation of next } \mathbf{e}\end{matrix}$$

Figure: Overview of Firekite's design.

- Each error position requires $\log n$ bits, hence length of $\mathbf{c}_e$ is $k \cdot \log n$.

Design [BKLV21]

Assume $n \gg m$, and k are integers.



noisy product of length n

$$\overset{m}{\mathbf{v}} \cdot \overset{m \times n}{\mathbf{M}} + \overset{n}{\mathbf{e}} = \begin{pmatrix} \mathbf{g} \\ \mathbf{v}' \\ \mathbf{c}_e \end{pmatrix} \begin{matrix} \longrightarrow \text{output} \\ \longrightarrow \text{next } \mathbf{v} \\ \longrightarrow \text{concise presentation of next } \mathbf{e} \end{matrix}$$

error vector of weight k

Figure: Overview of Firekite's design.

- Each error position requires $\log n$ bits, hence length of $\mathbf{c}_e$ is $k \cdot \log n$.
- Keystream length is $d = n - m - k \cdot \log n$.

Design [BKLV21]

Assume $n \gg m$, and k are integers.

$$\overset{m}{\mathbf{v}} \cdot \overset{m \times n}{\mathbf{M}} + \underset{\text{error vector of weight } k}{\overset{n}{\mathbf{e}}} = \overset{\text{noisy product of length } n}{\begin{pmatrix} \mathbf{g} \\ \mathbf{v}' \\ \mathbf{c}_e \end{pmatrix}} \begin{matrix} \longrightarrow \text{output} \\ \longrightarrow \text{next } \mathbf{v} \\ \longrightarrow \text{concise presentation of next } \mathbf{e} \end{matrix}$$

Figure: Overview of Firekite's design.

- Each error position requires $\log n$ bits, hence length of $\mathbf{c}_e$ is $k \cdot \log n$.
- Keystream length is $d = n - m - k \cdot \log n$.
- For efficiency, the authors proposed using a 'cyclic' **M**.

The Learning Parity with Noise Problem

LPN oracle.

Let $\mathbf{x} \xleftarrow{\$} \{0,1\}^m$ and $\eta \in (0, \frac{1}{2})$. An LPN *oracle* Π_{LPN} for $\mathbf{x}$ and η returns pairs of the form

$$\left(\mathbf{g} \xleftarrow{U} \{0,1\}^m, \langle \mathbf{x},\mathbf{g} \rangle \oplus e\right),$$

where $e \leftarrow \mathrm{Ber}_\eta$, and $\langle \mathbf{x}, \mathbf{g} \rangle$ denotes the scalar product of vectors $\mathbf{x}$ and $\mathbf{g}$.

The Learning Parity with Noise Problem

LPN oracle.

Let $\mathbf{x} \overset{\$}{\leftarrow} \{0,1\}^m$ and $\eta \in (0, \frac{1}{2})$. An LPN *oracle* Π_{LPN} for $\mathbf{x}$ and η returns pairs of the form

$$\left(\mathbf{g} \overset{U}{\leftarrow} \{0,1\}^m, \langle \mathbf{x},\mathbf{g} \rangle \oplus e\right),$$

where $e \leftarrow \mathsf{Ber}_\eta$, and $\langle \mathbf{x}, \mathbf{g} \rangle$ denotes the scalar product of vectors $\mathbf{x}$ and $\mathbf{g}$.

LPN problem, Search version, informal.

Given an LPN oracle Π_{LPN} with parameters m and η. The (m, η)-LPN problem is finding the secret vector $\mathbf{x}$ from observing N samples from (m, η)-Π_{LPN} oracle.

ctd.

- Search version $\overset{\textit{polynomial}}{\sim}$ Decision version [KS06].

ctd.

- Search version $\overset{\textit{polynomial}}{\sim}$ Decision version [KS06].
- We can rewrite $\mathbf{g}_1, \ldots, \mathbf{g}_N$ as a matrix $\mathbf{G}$, and the LPN problem becomes finding $\mathbf{x}$ given its noisy product with $\mathbf{G}$.

$$\mathbf{xG} + \mathbf{e}$$

ctd.

- Search version $\overset{\textit{polynomial}}{\sim}$ Decision version [KS06].
- We can rewrite $\mathbf{g}_1, \ldots, \mathbf{g}_N$ as a matrix $\mathbf{G}$, and the LPN problem becomes finding $\mathbf{x}$ given its noisy product with $\mathbf{G}$.

$$\mathbf{xG} + \mathbf{e}$$

Remark:

It is closely related to the *Syndrome Decoding Problem.*

Firekite vs LPN

- The error in Firekite is of weight at most k vs. Bernoulli distribution.

Firekite vs LPN

- The error in Firekite is of weight at most k vs. Bernoulli distribution.
- Less information with Firekite.
- Cyclic **M** (Ring-LPN variant) vs. uniformly random **G**.

Security:

Firekite vs LPN

- The error in Firekite is of weight at most k vs. Bernoulli distribution.
- Less information with Firekite.
- Cyclic $\mathbf{M}$ (Ring-LPN variant) vs. uniformly random $\mathbf{G}$.

Security:

- Reduction to LPN.

Firekite vs LPN

- The error in Firekite is of weight at most k vs. Bernoulli distribution.
- Less information with Firekite.
- Cyclic **M** (Ring-LPN variant) vs. uniformly random **G**.

Security:

- Reduction to LPN.
- Apply cryptanalysis methods to the (corresponding) LPN-instance.

Assumption:

Firekite vs LPN

- The error in Firekite is of weight at most k vs. Bernoulli distribution.
- Less information with Firekite.
- Cyclic $\mathbf{M}$ (Ring-LPN variant) vs. uniformly random $\mathbf{G}$.

Security:

- Reduction to LPN.
- Apply cryptanalysis methods to the (corresponding) LPN-instance.

Assumption:

- Ring-LPN is secure.

Firekite vs LPN

- The error in Firekite is of weight at most k vs. Bernoulli distribution.
- Less information with Firekite.
- Cyclic $\mathbf{M}$ (Ring-LPN variant) vs. uniformly random $\mathbf{G}$.

Security:

- Reduction to LPN.
- Apply cryptanalysis methods to the (corresponding) LPN-instance.

Assumption:

- Ring-LPN is secure.
- A Firekite instance is as hard as its (corresponding) LPN-instance.

Distinguishing Attacks

Key Observations

$$\mathbf{v}_i \cdot \mathbf{M}_{[d]} + \mathbf{e}_i = \mathbf{g}_i.$$

Observation 1

The first d columns $\mathbf{M}_{[d]}$ is fixed.

Key Observations

$$\mathbf{v}_i \cdot \mathbf{M}_{[d]} + \mathbf{e}_i = \mathbf{g}_i.$$

Observation 1

The first d columns $\mathbf{M}_{[d]}$ is fixed.

Observation 2

Key Observations

$$\mathbf{v}_i \cdot \mathbf{M}_{[d]} + \mathbf{e}_i = \mathbf{g}_i.$$

Observation 1

The first d columns $\mathbf{M}_{[d]}$ is fixed.

Observation 2

- If $\sum_{i=1}^{\ell} \mathbf{v}_i = \mathbf{0}$, then $\sum_{i=1}^{\ell} \mathbf{g}_i = \sum_{i=1}^{\ell} \mathbf{e}_i$. Moreover, $\mathbf{e}_i$ is sparse ($k \ll n$).

Key Observations

$$\mathbf{v}_i \cdot \mathbf{M}_{[d]} + \mathbf{e}_i = \mathbf{g}_i.$$

Observation 1

The first d columns $\mathbf{M}_{[d]}$ is fixed.

Observation 2

- If $\sum_{i=1}^{\ell} \mathbf{v}_i = \mathbf{0}$, then $\sum_{i=1}^{\ell} \mathbf{g}_i = \sum_{i=1}^{\ell} \mathbf{e}_i$. Moreover, $\mathbf{e}_i$ is sparse ($k \ll n$).
- Since $m < d$, we expect to see low Hamming-weight combinations more frequently than the random case.

How to exploit the observations?

We only see this!

$$\mathbf{v}_i \cdot \mathbf{M}_{[d]} + \mathbf{e}_i = \boxed{\mathbf{g}_i}.$$

Figure: Applying our ideas with observing the keystream.

How to exploit the observations?

We only see this!

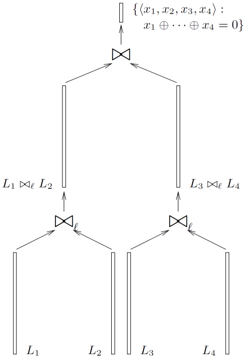
$$\mathbf{v}_i \cdot \mathbf{M}_{[d]} + \mathbf{e}_i = \mathbf{g}_i.$$

Figure: Applying our ideas with observing the keystream.

Idea:

If we can detect a low-weight sum of $\mathbf{g}_i$, and it is statistically implausible to have such a sum in random case, then it must have come from a collision in $\mathbf{v}_i$.

How to efficiently detect low-weight sums?

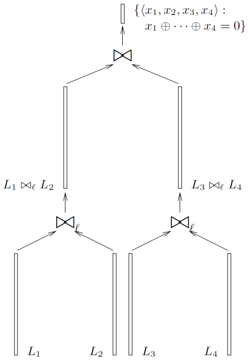


A pictorial representation of our algorithm for the 4-sum problem.

Figure: Match-and-Filter [BM17]

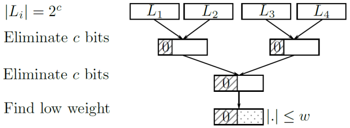
Figure: Wagner algorithm [Wag02]

How to efficiently detect low-weight sums?



A pictorial representation of our algorithm for the 4-sum problem.

Figure: Wagner algorithm [Wag02]



Match-and-Filter algorithm for $k = 4$

Figure: Match-and-Filter [BM17]

Our algorithm

We apply essential ideas and arguments from the two above algorithms, with some flavor from BKW-algorithm to detect low-weight ℓ-sums.

Modifications for our algorithm

- Instead of ℓ lists, we use only 1 initial list $L^{(0)}$, with an increased size. In particular, to cancel c bits and maintain the list size, $L^{(0)} \approx 3 \cdot 2^{c}$.

Our algorithm

We apply essential ideas and arguments from the two above algorithms, with some flavor from BKW-algorithm to detect low-weight ℓ-sums.

Modifications for our algorithm

- Instead of ℓ lists, we use only 1 initial list $L^{(0)}$, with an increased size. In particular, to cancel c bits and maintain the list size, $L^{(0)} \approx 3 \cdot 2^c$.
- We call COMBINE the routine to find vectors that collide in c bits. Let $t = \log \ell$, we need to apply COMBINE t times, resulting in $L^{(0)} \longrightarrow L^{(1)} \cdots \longrightarrow L^{(t)}$.

Our algorithm

We apply essential ideas and arguments from the two above algorithms, with some flavor from BKW-algorithm to detect low-weight ℓ-sums.

Modifications for our algorithm

- Instead of ℓ lists, we use only 1 initial list $L^{(0)}$, with an increased size. In particular, to cancel c bits and maintain the list size, $L^{(0)} \approx 3 \cdot 2^c$.
- We call COMBINE the routine to find vectors that collide in c bits. Let $t = \log \ell$, we need to apply COMBINE t times, resulting in $L^{(0)} \longrightarrow L^{(1)} \cdots \longrightarrow L^{(t)}$.
- The parameter c in our algorithms needs to be bigger than in Wagner's algorithm. The reason is, we also need the observed g_i to be at least *error-free* modulo 2 in $t \cdot c$ positions.

Our algorithm, Combine.

- We can use *c* tuples as indices/keys in a hash table and detect collisions in each iteration.

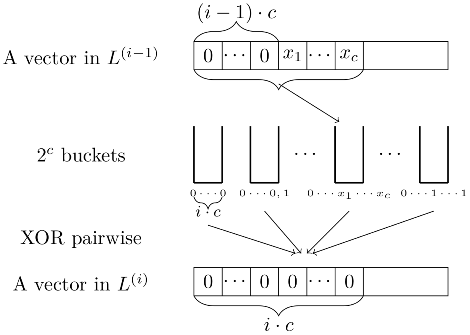
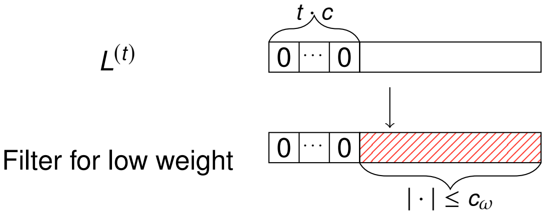


Figure 1: Combine for $L^{(i-1)}$.

Our algorithm, Filter



$L^{(t)}$

Filter for low weight

Figure: Filter $L^{(t)}$ with c_ω.

Remaining questions

- Q: How do we set the target weight c_ω?

Remaining questions

- Q: How do we set the target weight c_ω?
- A: Heuristically, $c_\omega \approx \frac{\ell \cdot k \cdot d}{n}$. Recall $k = \omega_H(\mathbf{e}_i)$ and d is the length of each $\mathbf{g}_i$.

Remaining questions

- Q: How do we set the target weight c_ω?
- A: Heuristically, $c_\omega \approx \frac{\ell \cdot k \cdot d}{n}$. Recall $k = \omega_H(\mathbf{e}_i)$ and d is the length of each $\mathbf{g}_i$.
- Q: How large is ℓ, i.e., algorithmic steps $t = \log \ell$?

Remaining questions

- Q: How do we set the target weight c_ω?
- A: Heuristically, $c_\omega \approx \frac{\ell \cdot k \cdot d}{n}$. Recall $k = \omega_H(\mathbf{e}_i)$ and d is the length of each $\mathbf{g}_i$.

- Q: How large is ℓ, i.e., algorithmic steps $t = \log \ell$?
- A: $\ell = 4, 8$ is reasonable (for most Firekite parameters).

Remaining questions

- Q: How do we set the target weight c_ω?
- A: Heuristically, $c_\omega \approx \frac{\ell \cdot k \cdot d}{n}$. Recall $k = \omega_H(\mathbf{e}_i)$ and d is the length of each $\mathbf{g}_i$.

- Q: How large is ℓ, i.e., algorithmic steps $t = \log \ell$?
- A: $\ell = 4, 8$ is reasonable (for most Firekite parameters).

- Q: How many vectors, i.e., $L^{(0)} = 3 \cdot 2^c$, do we need?

Remaining questions

- Q: How do we set the target weight c_ω?
- A: Heuristically, $c_\omega \approx \frac{\ell \cdot k \cdot d}{n}$. Recall $k = \omega_H(\mathbf{e}_i)$ and d is the length of each $\mathbf{g}_i$.

- Q: How large is ℓ, i.e., algorithmic steps $t = \log \ell$?
- A: $\ell = 4, 8$ is reasonable (for most Firekite parameters).

- Q: How many vectors, i.e., $L^{(0)} = 3 \cdot 2^c$, do we need?
- A: Assume P_{nf} is defined as the probability a low-weight sum is error-free modulo 2 in the first tc bits. If Wagner algorithm requires c, we need an overhead $\alpha(P_{nf})$, so $c + \alpha(P_{nf})$.

Analysis

Memory (the exponent c).

Recall:

$$\mathbf{v_i} \cdot \mathbf{M}_{[d]} = \mathbf{g}_i.$$

Memory (the exponent c).

Recall:

$$\mathbf{v_i} \cdot \mathbf{M}_{[d]} = \mathbf{g}_i.$$

Parameters: $\ell, t = \log \ell, m, d, c_\omega$.

Memory (the exponent c).

Recall:

$$\mathbf{v_i} \cdot \mathbf{M}_{[d]} = \mathbf{g}_i.$$

Parameters: $\ell, t = \log \ell, m, d, c_\omega$.

A collision of ℓ length-m vectors $\mathbf{v_i}$, according to Wagner, requires $2^{\frac{m}{1+\log \ell}}$, so we need $2^{\frac{m}{1+\log \ell}+\alpha(P_{nf})}$.

Memory (the exponent c).

Recall:

$$\mathbf{v_i} \cdot \mathbf{M}_{[d]} = \mathbf{g}_i.$$

Parameters: $\ell, t = \log \ell, m, d, c_\omega$.

A collision of ℓ length-m vectors $\mathbf{v_i}$, according to Wagner, requires $2^{\frac{m}{1+\log \ell}}$, so we need $2^{\frac{m}{1+\log \ell}+\alpha(P_{nf})}$.

For P_{nf}, we can rely on a lower bound. In particular, $P_{nf} \geq$ the probability that all errors $\mathbf{e}$ are zeros at the first $t \cdot c$ positions.

Memory (the exponent c).

Recall:

$$\mathbf{v_i} \cdot \mathbf{M}_{[d]} = \mathbf{g}_i.$$

Parameters: $\ell, t = \log \ell, m, d, c_\omega$.

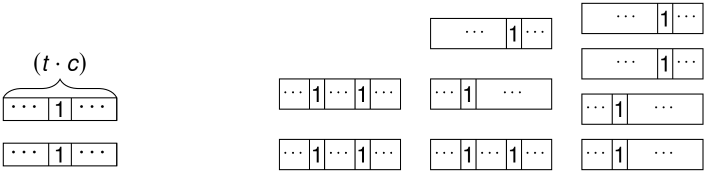
A collision of ℓ length-m vectors $\mathbf{v_i}$, according to Wagner, requires $2^{\frac{m}{1+\log \ell}}$, so we need $2^{\frac{m}{1+\log \ell}+\alpha(P_{nf})}$.

For P_{nf}, we can rely on a lower bound. In particular, $P_{nf} \geq$ the probability that all errors **e** are zeros at the first $t \cdot c$ positions.

Remark

- The better we ‘estimate’ P_{nf}, the smaller $\alpha(P_{nf})$ is.
- For $\ell = 8$, we consider more complicated error patterns in the first $t \cdot c$ bits.

Some examples of the error colliding patterns in canceled bits.



(a) 1 double error

(b) 2 double errors

Figure: Illustration for the colliding patterns

Summary of how to choose c:

Let N be the number of low-weight sums we ‘expect’ to find by our algorithm. Then

Summary of how to choose c:

Let N be the number of low-weight sums we ‘expect’ to find by our algorithm. Then For $\ell = 4$, we have

$$N = \binom{3 \cdot 2^c}{4} \cdot 2^{-m} \cdot 3 \cdot 2^{-c} \cdot P_{\text{nf}} > 1$$

Summary of how to choose c:

Let N be the number of low-weight sums we 'expect' to find by our algorithm. Then

For $\ell = 4$, we have

$$N = \binom{3 \cdot 2^c}{4} \cdot 2^{-m} \cdot 3 \cdot 2^{-c} \cdot P_{\mathrm{nf}} > 1$$

For $\ell = 8$, we have

$$N = \binom{3 \cdot 2^c}{8} \cdot 2^{-m} \cdot 105 \cdot 2^{-4c} \cdot P_{\mathrm{nf}} > 1$$

Complexity

$$C = t \cdot (3 \cdot 2^c) \cdot (1 + \lfloor d/p \rfloor).$$

On average, we have to do $3 \cdot 2^c$ XOR operations in each iteration of Combine. Each XOR cost $1 + \lfloor d/p \rfloor$, where p is the number of bits that can be XOR-ed in each operation.

Note

Of course, there are other algorithmic costs but this is the dominating part.

Success Probability

How to ‘interpret’ the low-weight sums that have been found?

The low-weight sums must be easily distinguished from those that can happen by sheer chances. In other words, it must be statistically improbable for such a low-weight sum to appear.

$$N_{\text{random}} = 3 \cdot 2^{c} \cdot \frac{\sum_{i=0}^{c_\omega} \binom{d-t\cdot c}{i}}{2^{d-t\cdot c}} \approx \sum_{i=0}^{c_\omega} 2^{-\left(1-H\left(\frac{i}{d-t\cdot c}\right)\right)(d-t\cdot c)+c} \approx 2^{-\left(1-H\left(\frac{c_\omega}{d-t\cdot c}\right)\right)(d-t\cdot c)+c}.$$

Results

Attacks on Firekite with different parameters.

Table: Our distinguishing attack complexity for 80-bit and 128-bit security of Firekite.

Parameters				Memory (c)		Time(log)		N_{random}(log)	
m	n	k	Security	4-sum	8-sum	4-sum	8-sum	4-sum	8-sum
216	1024	16	82.76	76	62	80.17	66.75	-215.76	-90.23
216	2048	32	82.76	76	62	81.17	67.75	-765.79	-465.74
216	16,384	216	80.68	75	60	83.28	68.87	-9011.62	-6541.71
352	2048	32	129.07	125	101	130.16	106.75	-541.40	-275.94
352	4096	58	128.95	124	99	130.17	105.75	-1739.41	-1150.69
352	16,384	228	128.93	123	99	131.26	107.84	-8510.19	-6023.39

Key Recovery (in prose)

How does ISD algorithm work?

Problem:

Let C be a random binary code generated by a matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$. Let $\mathbf{y} = \mathbf{c} + \mathbf{e}$ be a noisy codeword where $\mathbf{c} \in C$, and $\omega_H(\mathbf{e}) = \omega <$ minimum distance of C. Recover $\mathbf{y}$, or $\mathbf{e}$.

How does ISD algorithm work?

Problem:

Let $\mathcal{C}$ be a random binary code generated by a matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$. Let $\mathbf{y} = \mathbf{c} + \mathbf{e}$ be a noisy codeword where $\mathbf{c} \in \mathcal{C}$, and $\omega_H(\mathbf{e}) = \omega <$ minimum distance of $\mathcal{C}$. Recover $\mathbf{y}$, or $\mathbf{e}$.

$$\mathcal{C}' = \mathcal{C} + \mathbf{y} = \langle \begin{pmatrix} \mathbf{G} \\ \mathbf{y} \end{pmatrix} \rangle.$$

How does ISD algorithm work?

Problem:

Let $\mathcal{C}$ be a random binary code generated by a matrix $\mathbf{G} \in \mathbb{F}_2^{k\times n}$. Let $\mathbf{y} = \mathbf{c} + \mathbf{e}$ be a noisy codeword where $\mathbf{c} \in \mathcal{C}$, and $\omega_H(\mathbf{e}) = \omega <$ minimum distance of $\mathcal{C}$. Recover $\mathbf{y}$, or $\mathbf{e}$.

$$\mathcal{C}' = \mathcal{C} + \mathbf{y} = \langle \begin{pmatrix} \mathbf{G} \\ \mathbf{y} \end{pmatrix} \rangle.$$

Equivalently, we find a codeword of weight ω of this new code.

How does ISD algorithm work?

Problem:

Let $\mathcal{C}$ be a random binary code generated by a matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$. Let $\mathbf{y} = \mathbf{c} + \mathbf{e}$ be a noisy codeword where $\mathbf{c} \in \mathcal{C}$, and $\omega_H(\mathbf{e}) = \omega <$ minimum distance of $\mathcal{C}$. Recover $\mathbf{y}$, or $\mathbf{e}$.

$$\mathcal{C}' = \mathcal{C} + \mathbf{y} = \langle \begin{pmatrix} \mathbf{G} \\ \mathbf{y} \end{pmatrix} \rangle .$$

Equivalently, we find a codeword of weight ω of this new code.

$$\begin{pmatrix} \mathbf{G} \\ \mathbf{y} \end{pmatrix} \xrightarrow{Gauss \circ \pi} \begin{pmatrix} \mathbf{I} & \mathbf{J} \end{pmatrix}$$

How does ISD algorithm work?

Problem:

Let C be a random binary code generated by a matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$. Let $\mathbf{y} = \mathbf{c} + \mathbf{e}$ be a noisy codeword where $\mathbf{c} \in C$, and $\omega_H(\mathbf{e}) = \omega <$ minimum distance of C. Recover $\mathbf{y}$, or $\mathbf{e}$.

$$C' = C + \mathbf{y} = \langle \begin{pmatrix} \mathbf{G} \\ \mathbf{y} \end{pmatrix} \rangle.$$

Equivalently, we find a codeword of weight ω of this new code.

$$\begin{pmatrix} \mathbf{G} \\ \mathbf{y} \end{pmatrix} \xrightarrow{Gauss \circ \pi} \begin{pmatrix} \mathbf{I} & \mathbf{J} \end{pmatrix}$$

Assume the first part is of dimension k, we run through weight p vectors $\mathbf{u}$ of length k and check for $\omega_H(\mathbf{uJ}) = \omega - p$.

Applying our distinguishing attack.

- First, use our distinguisher to collect N low-weight 8-sums.

Applying our distinguishing attack.

- First, use our distinguisher to collect N low-weight 8-sums.
- For each sum, we get 7 independent vectors $\Rightarrow 7N > m$.

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1 \\ \cdots \\ \mathbf{g}_{7N} \end{pmatrix}$$

Applying our distinguishing attack.

- First, use our distinguisher to collect N low-weight 8-sums.
- For each sum, we get 7 independent vectors $\Rightarrow 7N > m$.

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1 \\ \cdots \\ \mathbf{g}_{7N} \end{pmatrix}$$

- Looking at each sum, we can remove all ‘direct error contributions’, and we only keep columns where all low-weight sums are zero (at that corresponding bit).

Applying our distinguishing attack.

- First, use our distinguisher to collect N low-weight 8-sums.
- For each sum, we get 7 independent vectors $\Rightarrow 7N > m$.

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1 \\ \cdots \\ \mathbf{g}_{7N} \end{pmatrix}$$

- Looking at each sum, we can remove all 'direct error contributions', and we only keep columns where all low-weight sums are zero (at that corresponding bit).
- Whatever left is a much smaller matrix where double errors are still present.

Applying our distinguishing attack.

- First, use our distinguisher to collect N low-weight 8-sums.
- For each sum, we get 7 independent vectors $\Rightarrow 7N > m$.

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1 \\ \cdots \\ \mathbf{g}_{7N} \end{pmatrix}$$

- Looking at each sum, we can remove all ‘direct error contributions’, and we only keep columns where all low-weight sums are zero (at that corresponding bit).
- Whatever left is a much smaller matrix where double errors are still present.
- When recombined, these double errors yield a small weight codeword.

Applying our distinguishing attack.

- First, use our distinguisher to collect N low-weight 8-sums.
- For each sum, we get 7 independent vectors $\Rightarrow 7N > m$.

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1 \\ \cdots \\ \mathbf{g}_{7N} \end{pmatrix}$$

- Looking at each sum, we can remove all ‘direct error contributions’, and we only keep columns where all low-weight sums are zero (at that corresponding bit).
- Whatever left is a much smaller matrix where double errors are still present.
- When recombined, these double errors yield a small weight codeword.
- Estimate the double errors, then apply ISD algorithms.

Conclusions

- Better security measures of Firekite.

Conclusions

- Better security measures of Firekite.
- LPN-based constructions are interesting (provable security, efficient).

Conclusions

- Better security measures of Firekite.
- LPN-based constructions are interesting (provable security, efficient).
- Fixes.

Conclusions

- Better security measures of Firekite.
- LPN-based constructions are interesting (provable security, efficient).
- Fixes.
- Further works.

References

Sonia Bogos, Dario Korolija, Thomas Locher, and Serge Vaudenay.
[BKLV21] Towards efficient lpn-based symmetric encryption.

Leif Both and Alexander May.
[BM17] The approximate k-list problem.

Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman.
[HDWH12] Mining your ps and qs: Detection of widespread weak keys in network devices.

Jonathan Katz and Ji Sun Shin.
[KS06] Parallel and concurrent security of the HB and hb$^+$ protocols.

Peter W. Shor.
[Sho99] Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.

David A. Wagner.
[Wag02] A generalized birthday problem.

LUND
UNIVERSITY