Never trust a bunny

D. J. Bernstein
University of Illinois at Chicago

Tanja Lange
Technische Universiteit Eindhoven

The HB(n, τ, τ') protocol

(2001 Hopper–Blum)

Secret $s \in \mathbf{F}_2^n$.

RFID reader sends random $C \in \mathbf{F}_2^{n \times n}$.

Tag sends $T = Cs + e$
where each bit of e is
set with probability τ.

Reader checks that
$T - Cs$ has $\leq \tau' n$ bits set.

"Reasonable" parameters:
$n = 512$, $\tau = 1/8$, $\tau' = 1/4$.

The LPN(n, τ) problem

Computational LPN problem:
compute s given
random R_1; $R_1 s + e_1$;
random R_2; $R_2 s + e_2$; ...

Equivalently: Compute s given
random $r_1 \in \mathbf{F}_2^n$; $r_1 \cdot s + e_1$;
random $r_2 \in \mathbf{F}_2^n$; $r_2 \cdot s + e_2$; ...

Solving computational LPN
breaks HB and all of the
other protocols in this talk.

(Warning: “The LPN problem”
is normally defined as
a decisional problem.)

Breaking HB without solving LPN

Attacker sends to the tag:

$$C = \begin{pmatrix} 1\,0\,0\ldots0 \\ 1\,0\,0\ldots0 \\ 1\,0\,0\ldots0 \\ \vdots\ \vdots\ \vdots\ \ddots\ \vdots \\ 1\,0\,0\ldots0 \end{pmatrix}.$$

Majority vote of tag response
is very likely to be
first bit of s.
Repeat for other bits.

Many subsequent HB variants
try to resist active attacks.

MatrixLapin(n, τ, τ')

Secrets $s, s' \in \mathbf{F}_2^n$.

Reader sends random $C \in \mathbf{F}_2^{n \times n}$.
(Improvement: restrict to "nice" subspace; same in next protocol.)

Tag sends
random invertible $R \in \mathbf{F}_2^{n \times n}$
and $T = R(Cs + s') + e$
where each bit of e is
set with probability τ.

Reader checks that
R is invertible and that
$T - R(Cs + s')$ has $\le \tau' n$ bits set.

Lapin(n, f, τ, τ') where $\deg f = n$

(FSE 2012 Heyse–Kiltz–Lyubashevsky–Paar–Pietrzak)

Secrets $s, s' \in \mathbf{F}_2[x]/f$.

Reader sends random $c \in \mathbf{F}_2[x]/f$.

Tag sends
random invertible $r \in \mathbf{F}_2[x]/f$
and $t = r(cs + s') + e$
where each bit of e is
set with probability τ.

Reader checks that
r is invertible and that
$t - r(cs + s')$ has $\leq \tau' n$ bits set.

Ring-LPN(n, f, τ)

Lapin c and r correspond to matrices C and R.
Highly non-random matrices!
Saves space and time but maybe risks attacks.

Computational Ring-LPN problem (FSE 2012): compute s given random r_1; $r_1 s + e_1$; random r_2; $r_2 s + e_2$; $\ldots$

Feed c repeatedly to Lapin tag, solve Ring-LPN $\Rightarrow cs + s'$.
Repeat with c' where $c - c'$ is invertible, obtain s and s'.

2000 Blum–Kalai–Wasserman

Standard attack on LPN.

Main idea: If r_1 and r_2 have the same starting bits then $r_1 + r_2$ has starting bits 0 and $t_1 + t_2 = (r_1 + r_2) \cdot s + (e_1 + e_2)$.

Repeat: clear more bits, obtain $(0, 0, \ldots, 0, 1)$ as a combination of 2^a values r_i. Corresponding t combination is last bit of s with noise.

Use many combinations to eliminate noise.

2006 Levieil–Fouque

Same main idea,
but clear fewer bits.
Obtain $(0, 0, \ldots, 0, *, \ldots, *)$
for every pattern of $*, \ldots, *$.

Enumerate each possibility
for bits of s at $*$ positions.
Use fast Walsh transform.

Advantage: smaller noise.
Need fewer queries, less memory,
less computation.

With this, attacks on Lapin were
claimed to need at least "2^{77}
memory".

2006 Levieil–Fouque

Same main idea,
but clear fewer bits.
Obtain $(0, 0, \ldots, 0, *, \ldots, *)$
for every pattern of $*, \ldots, *$.

Enumerate each possibility
for bits of s at $*$ positions.
Use fast Walsh transform.

Advantage: smaller noise.
Need fewer queries, less memory,
less computation.

With this, attacks on Lapin were
claimed to need at least "2^{77}
memory". Actually $\approx 2^{82}$ bytes.

2011 Kirchner

Assume matrix R_1 is invertible.
Compute R_1^{-1} and
$R_2 R_1^{-1}(R_1 s + e_1) + R_2 s + e_2$,
$R_3 R_1^{-1}(R_1 s + e_1) + R_3 s + e_3$,
$R_4 R_1^{-1}(R_1 s + e_1) + R_4 s + e_4$, . . .

2011 Kirchner

Assume matrix R_1 is invertible.
Compute R_1^{-1} and
$R_2 R_1^{-1}(R_1 s + e_1) + R_2 s + e_2$,
$R_3 R_1^{-1}(R_1 s + e_1) + R_3 s + e_3$,
$R_4 R_1^{-1}(R_1 s + e_1) + R_4 s + e_4$, ...

Obtain new LPN(n, τ) problem
R'_2; $R'_2 e_1 + e_2$;
R'_3; $R'_3 e_1 + e_3$;
R'_4; $R'_4 e_1 + e_4$; ...

2011 Kirchner

Assume matrix R_1 is invertible.
Compute R_1^{-1} and
$R_2 R_1^{-1}(R_1 s + e_1) + R_2 s + e_2$,
$R_3 R_1^{-1}(R_1 s + e_1) + R_3 s + e_3$,
$R_4 R_1^{-1}(R_1 s + e_1) + R_4 s + e_4$, ...

Obtain new LPN(n, τ) problem
R'_2; $R'_2 e_1 + e_2$;
R'_3; $R'_3 e_1 + e_3$;
R'_4; $R'_4 e_1 + e_4$; ...
with sparse secret e_1.

Guess some bits of e_1,
cancel fewer bits;
less noise to deal with.

Our attack on Lapin

Main improvements in paper:

- Use the ring structure to save time in computations.
- Better guessing strategy.

We break Ring-LPN$(512, 1/8)$ in
$<2^{56}$ bytes of memory,
$<2^{38}$ queries, and
$<2^{98}$ bit operations.

Our attack on Lapin

Main improvements in paper:

- Use the ring structure to save time in computations.
- Better guessing strategy.

We break Ring-LPN$(512, 1/8)$ in
$<2^{56}$ bytes of memory,
$<2^{38}$ queries, and
$<2^{98}$ bit operations.

Many tradeoffs possible: e.g.,
$<2^{78}$ bytes of memory,
$<2^{63}$ queries, and
$<2^{88}$ bit operations.

What about LPN?

Better guessing strategy
also helps for LPN.

We break LPN$(1024, 1/20)$ in
$<2^{21}$ bytes of memory,
$<2^{64}$ queries, and
$<2^{100}$ bit operations
(or $<2^{93}$ for Ring-LPN).

Also have a new trick
to reduce # queries.
LPN$(1024, 1/20)$: 10 queries!

Picture taken 2012.04.27 at CWI: