NISTPQC is so much fun! 10900qmmP

Lorenz Panny

Technische Universiteit Eindhoven

Costa Adeje, Tenerife, 31 January 2018
Not a competition.

69 submissions published on December 21st 2017.

First complete break: 3 hours later.

(slide stolen from Daniel J. Bernstein)

(slide stolen from Daniel J. Bernstein)
How to find a good victim?

The optimal target...
How to find a good victim?

The optimal target...

- is a construction you’ve never heard of.
How to find a good victim?

The optimal target...

- is a construction you’ve never heard of.
- was typeset in *Microsoft Word* (or similar).
How to find a good victim?

The optimal target...

- is a construction you’ve never heard of.
- was typeset in **Microsoft Word** (or similar).
- makes completely **wild security claims**.
UNCONDITIONALLY SECURE PUBLIC-KEY ENCRYPTION

— ‘Guess Again’ NISTPQC submission document
UNCONDITIONALLY SECURE PUBLIC-KEY ENCRYPTION

It is well known (and easy to show) that unconditionally secure [...] public-key encryption is impossible

— ‘Guess Again’ NISTPQC submission document
It is well known (and easy to show) that unconditionally secure [...] public-key encryption is impossible if the legitimate receiver decrypts correctly with probability exactly 1. The question is: what if this probability is less than 1?

— ‘Guess Again’ NISTPQC submission document
UNCONDITIONALLY SECURE PUBLIC-KEY ENCRYPTION (WITH POSSIBLE DECRYPTION ERRORS)

It is well known (and easy to show) that unconditionally secure [...] public-key encryption is impossible if the legitimate receiver decrypts correctly with probability exactly 1. The question is: what if this probability is less than 1?

— ‘Guess Again’ NISTPQC submission document
[...] legitimate sender has an advantage over the eavesdropper since the sender [...] knows exactly what the transmitted secret bit is. [...] instead of making the receiver guess the transmitted bit we make the sender guess the receiver’s decryption key [...].

— ‘Guess Again’ NISTPQC submission document
Guess Again

Public parameters: $n = 256, h = 2000, g = 2000, f = 120000$. 
Guess Again

Public parameters: \( n = 256, h = 2000, g = 2000, f = 120000 \).

Bob:

- Pick a private key \( b \) \(\leftarrow\) \{0 \ldots n - 1\}.
- Let public key \( B := \text{random\_walk}(b, h) \). Restart if \( B \geq n \).
Guess Again

Public parameters: \( n = 256, h = 2000, g = 2000, f = 120000 \).

Bob:
- Pick a private key \( b \leftarrow \text{random} \{0...n-1\} \).
- Let public key \( B := \text{random\_walk}(b, h) \). Restart if \( B \geq n \).

Alice transmits a single bit \( m \in \{0, 1\} \) to Bob:
- Select \( \lessdot \leftarrow \text{random} \{<, >\} \) and \( s \leftarrow \text{random} \{f, g\} \).
- Pick \( a \leftarrow \text{random} \{B...n-1\} \) and let \( A := \text{random\_walk}(a, s) \). Repeat this until \( A \lessdot B \).
- If \( (\lessdot, s) \in \{(>, f), (<, g)\} \), send \((m, a)\). Else send \((1 \oplus m, a)\).

Repeat many times to increase success probability.
Guess Again

Public parameters: $n = 256, h = 2000, g = 2000, f = 120000$.

Bob:

- Pick a private key $b \leftarrow \text{random} \{0...n - 1\}$.
- Let public key $B := \text{random}_\text{walk}(b, h)$. Restart if $B \geq n$.

Alice transmits a single bit $m \in \{0, 1\}$ to Bob:

- Select $\ll \\text{random} \{<, >\}$ and $s \leftarrow \text{random} \{f, g\}$.
- Pick $a \leftarrow \text{random} \{B...n - 1\}$ and let $A := \text{random}_\text{walk}(a, s)$. Repeat this until $A \ll B$.
- If $(\ll, s) \in \{(>, f), (<, g)\}$, send $(m, a)$. Else send $(1 \oplus m, a)$. Repeat many times to increase success probability.

Argument: Bit $m$ is flipped with probability $\frac{1}{2}$, thus secure. (?)
Guess Again

\[ n = 256, g = 2000, f = 120000. \]

- Pick \( a \leftarrow \text{random} \{ B \ldots n - 1 \} \) and let \( A := \text{random_walk}(a, s) \). Repeat this until \( A \leq B \).
- If \((\leq, s) \in \{(>, f), (<, g)\}\), send \((m, a)\). Else send \((1 \oplus m, a)\).
\( n = 256, \ g = 2000, \ f = 120000. \)

- Pick \( a \leftarrow \text{random} \ \{B \ldots n - 1\} \) and let \( A := \text{random\_walk}(a, s) \). Repeat this until \( A \leq B \).
- If \( (\leq, s) \in \{(>, f), (<, g)\} \), send \( (m, a) \). Else send \( (1 \oplus m, a) \).

Distribution of \( a \) conditional on \( (\leq, s) \):

<table>
<thead>
<tr>
<th>( A )</th>
<th>( s )</th>
<th>( a )</th>
<th>flip?</th>
</tr>
</thead>
<tbody>
<tr>
<td>( &lt; B )</td>
<td>( f )</td>
<td>slightly biased towards ( B )</td>
<td>0</td>
</tr>
<tr>
<td>( &lt; B )</td>
<td>( g )</td>
<td>strongly biased towards ( B )</td>
<td>1</td>
</tr>
<tr>
<td>( &gt; B )</td>
<td>( f )</td>
<td>more or less random</td>
<td>1</td>
</tr>
<tr>
<td>( &gt; B )</td>
<td>( g )</td>
<td>slightly biased away from ( B )</td>
<td>0</td>
</tr>
</tbody>
</table>
Guess Again

\[ n = 256, \ g = 2000, \ f = 120000. \]

- Pick \( a \leftarrow^{\text{random}} \{ B \ldots n - 1 \} \) and let \( A := \text{random\_walk}(a, s) \). Repeat this until \( A \preceq B \).
- If \((\preceq, s) \in \{(>, f), (<, g)\}\), send \((m, a)\). Else send \((1 \oplus m, a)\).

Distribution of \( a \) conditional on \((\preceq, s)\):

<table>
<thead>
<tr>
<th>( A )</th>
<th>( s )</th>
<th>( a )</th>
<th>flip?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt; B)</td>
<td>( f )</td>
<td>slightly biased towards ( B )</td>
<td>0</td>
</tr>
<tr>
<td>(&lt; B)</td>
<td>( g )</td>
<td>strongly biased towards ( B )</td>
<td>1</td>
</tr>
<tr>
<td>(&gt; B)</td>
<td>( f )</td>
<td>more or less random</td>
<td>1</td>
</tr>
<tr>
<td>(&gt; B)</td>
<td>( g )</td>
<td>slightly biased away from ( B )</td>
<td>0</td>
</tr>
</tbody>
</table>

\(\Rightarrow\) In expectation, \( a \) of ‘flipped’ ciphertexts are smaller.
def recover_bit(ct, bit):
    assert bit < len(ct) // 4000
    ts = [struct.unpack('BB', ct[i:i+2])
          for i in range(4000*bit, 4000*(bit+1), 2)]
    xs, ys = [a for a, b in ts if b == 1], [a for a, b in ts if b == 2]
    return sum(xs) / len(xs) >= sum(ys) / len(ys)

def decrypt(ct):
    res = sum(recover_bit(ct, b) << b
              for b in range(len(ct) // 4000))
    return int.to_bytes(res, len(ct) // 4000 // 8, 'little')
 [...] our protocol is IND-CCA2 secure against any passive adversary, even computationally unbounded one.

 [...] “Computationally unbounded” here includes all possible computers, *quantum or not.*

— ‘Guess Again’ NISTPQC submission document
• **Chebyshev polynomials:**

\[
T_0 = 1 \\
T_1 = t \\
T_n = 2t \cdot T_{n-1} - T_{n-2}
\]

• **Commutative semigroup under composition:**

\[
T_a \circ T_b = T_b \circ T_a = T_{a \cdot b}
\]

...screams Diffie-Hellman, but it is secure?
Fear not: for behold, I bring you good tidings of great joy!

Given $x$, $y = T_a(x)$, there is no efficient way to compute the secret parameter $a$ from $T_a(x)$, even if a quantum computer [...] is used for the attack, as will be shown.

— ‘RVB’ NISTPQC submission document
Alice            public            Bob

\[x \xleftarrow{\text{random}} [0; 1]\]
\[a \xleftarrow{\text{random}} \{0...10^{100}\}\]

\[x, T_a(x)\]

\[b \xleftarrow{\text{random}} \{0...10^{100}\}\]
\[T_b(x)\]

\[s := T_a(T_b(x))\]
\[s := T_b(T_a(x))\]

All computations using high-precision floating point numbers.
Trigonometry:\(^1\)

\[ \cos(n \cdot \alpha) = T_n(\cos \alpha) \quad \forall \alpha \in \mathbb{R}. \]

\(^1\)Fun fact: ‘cosine’ is an anagram of ‘so nice’.
Trigonometry:¹

\[ \cos(n \cdot \alpha) = T_n(\cos \alpha) \quad \forall \alpha \in \mathbb{R}. \]


\[ T_n(x) = y \]

\[ \iff \cos(n \cdot \arccos x) = y \]

¹Fun fact: ‘cosine’ is an anagram of ‘so nice’.
RVB

- **Trigonometry:**
  \[ \cos(n \cdot \alpha) = T_n(\cos \alpha) \quad \forall \alpha \in \mathbb{R}. \]

  \[ T_n(x) = y \]
  \[ \iff \cos(n \cdot \arccos x) = y \]
  \[ \iff \exists k \in \mathbb{Z}. \pm n \cdot \arccos x = \arccos y + 2k\pi \]

---

1 Fun fact: ‘cosine’ is an anagram of ‘so nice’.
RVB

Trigonometry:¹

\[ \cos(n \cdot \alpha) = T_n(\cos \alpha) \quad \forall \alpha \in \mathbb{R}. \]


\[ T_n(x) = y \]

\[ \iff \cos(n \cdot \arccos x) = y \]

\[ \iff \exists k \in \mathbb{Z}. \pm n \cdot \arccos x = \arccos y + 2k\pi \]

\[ \iff \exists k \in \mathbb{Z}. \pm n = \frac{\arccos y}{\arccos x} + k \cdot \frac{2\pi}{\arccos x} \]

¹Fun fact: ‘cosine’ is an anagram of ‘so nice’.
Trigonometry: \(^1\)

\[
\cos(n \cdot \alpha) = T_n(\cos \alpha) \quad \forall \alpha \in \mathbb{R}.
\]

Therefore \cite{Bergamo-D'Arco-DeSantis-Kocarev:2004}

\[
T_n(x) = y \\
\iff \cos(n \cdot \arccos x) = y \\
\iff \exists k \in \mathbb{Z}. \quad \pm n \cdot \arccos x = \arccos y + 2k\pi \\
\iff \exists k \in \mathbb{Z}. \quad \pm n = \frac{\arccos y}{\arccos x} + k \cdot \frac{2\pi}{\arccos x}
\]

Problem: Given \(\alpha, \beta \in \mathbb{R}\), find \(k \in \mathbb{Z}\) such that \(\alpha + k\beta \in \mathbb{Z}\).

\(^1\)Fun fact: ‘cosine’ is an anagram of ‘so nice’.
Problem: Given $\alpha, \beta \in \mathbb{R}$, find $k \in \mathbb{Z}$ such that $\alpha + k \beta \in \mathbb{Z}$.

- Consider equation over $\mathbb{R}/\mathbb{Z}$, i.e., only look at decimals:

$$\alpha + k \beta \equiv 0 \pmod{\mathbb{Z}}$$
Problem: Given $\alpha, \beta \in \mathbb{R}$, find $k \in \mathbb{Z}$ such that $\alpha + k\beta \in \mathbb{Z}$.

- Consider equation over $\mathbb{R}/\mathbb{Z}$, i.e., only look at decimals:

  $$\alpha + k\beta \equiv 0 \pmod{\mathbb{Z}}$$

- B–D’A–DS–K now implicitly assume $\alpha$ and $\beta$ are rational and multiply by $d = \text{lcm(denominators)}$:

  $$\alpha d + k \cdot \beta d \equiv 0 \pmod{d}$$

Easy using modular arithmetic. Problem solved?
Problem: Given $\alpha, \beta \in \mathbb{R}$, find $k \in \mathbb{Z}$ such that $\alpha + k\beta \in \mathbb{Z}$.

- Consider equation over $\mathbb{R}/\mathbb{Z}$, i.e., only look at decimals:

  $$\alpha + k\beta \equiv 0 \pmod{\mathbb{Z}}$$

- B–D’A–DS–K now implicitly assume $\alpha$ and $\beta$ are rational and multiply by $d = \text{lcm}(\text{denominators})$:

  $$\alpha d + k \cdot \beta d \equiv 0 \pmod{d}$$

  Easy using modular arithmetic. Problem solved?

- Extremely unrealistic: $\arccos(\mathbb{Q}) \cap \mathbb{Q} = \{0\}$.
  $\implies$ Rounding errors all over the place.
The conclusions of Bergamo et al. are mathematically wrong. [...] The methods to uncover the secret do definitely not work. A solvable diophantine equation derived from the inverse cosine function does not exist.

— ‘RVB’ NISTPQC submission document
The conclusions of Bergamo et al. are mathematically wrong. [...]  

The methods to uncover the secret do definitely not work. A solvable diophantine equation derived from the inverse cosine function does not exist. 

[...]  

Several hundred years ago everybody strongly believed that the sun rotates around the earth.  

— ‘RVB’ NISTPQC submission document
Let $\alpha, \beta \in \mathbb{R}$ and $k \in \mathbb{Z}$ such that $\alpha + k\beta \in \mathbb{Z}$.

Problem: Given approximations $a \approx \alpha$ and $b \approx \beta$, recover $k$. 
Let $\alpha, \beta \in \mathbb{R}$ and $k \in \mathbb{Z}$ such that $\alpha + k\beta \in \mathbb{Z}$.

Problem: Given approximations $a \approx \alpha$ and $b \approx \beta$, recover $k$. 

\[
\text{L}
\]
Let $\alpha, \beta \in \mathbb{R}$ and $k \in \mathbb{Z}$ such that $\alpha + k\beta \in \mathbb{Z}$.

Problem: Given approximations $a \approx \alpha$ and $b \approx \beta$, recover $k$. 
Let $\alpha, \beta \in \mathbb{R}$ and $k \in \mathbb{Z}$ such that $\alpha + k\beta \in \mathbb{Z}$.

Problem: Given approximations $a \approx \alpha$ and $b \approx \beta$, recover $k$. 
Let $K$ be an upper bound for $|k|$ and $B \approx K^2$. The lattice spanned by the rows of

\[
\begin{pmatrix}
B & 0 & \lfloor aB \rfloor \\
0 & 1 & \lfloor bB \rfloor \\
0 & 0 & B
\end{pmatrix}
\]

contains a short basis vector of the form $(B, k', \varepsilon)$ with $|\varepsilon| \ll B$. This corresponds to the relation

\[
\lfloor aB \rfloor + k' \lfloor bB \rfloor \equiv \varepsilon \pmod{B},
\]

and division by $B$ yields

\[
a + k'b \approx 0 \pmod{Z}.
\]
Let $K$ be an upper bound for $|k|$ and $B \approx K^2$. The lattice spanned by the rows of

$$\begin{pmatrix} B & 0 & \lfloor aB \rfloor \\ 0 & 1 & \lfloor bB \rfloor \\ 0 & 0 & B \end{pmatrix}$$

contains a short basis vector of the form

$$(B, k', \varepsilon)$$

with $|\varepsilon| \ll B$. \\

---

**RVB**
Let $K$ be an upper bound for $|k|$ and $B \approx K^2$. The lattice spanned by the rows of

$$
\begin{pmatrix}
B & 0 & \lfloor aB \rfloor \\
0 & 1 & \lfloor bB \rfloor \\
0 & 0 & B
\end{pmatrix}
$$

contains a short basis vector of the form $(B, k', \varepsilon)$ with $|\varepsilon| \ll B$. This corresponds to the relation

$$
\lfloor aB \rfloor + k' \lfloor bB \rfloor \equiv \varepsilon \pmod{B},
$$

and division by $B$ yields

$$
a + k'b \approx 0 \pmod{\mathbb{Z}}.
$$
def recover(pk):
    Tx, x = map(RealField(10**4), pk.split(\'\\0\')[2])

    a = arccos(Tx) / arccos(x)
    b = 2*pi / arccos(x)

    # find an integer k such that a + k * b is close to an integer
    B = 10**(len(pk)//2)
    M = matrix([[B, 0, round(a*B)], [0, 1, round(b*B)], [0, 0, B]])
    for l in M.LLL().rows():
        if l[0]:
            k = sign(l[0]) * l[1]
            break

    guess = abs(round(a + k * b))

    # brute-force a small range in case we are slightly off
    for d in range(256):
        for s in (-1, +1):
            if abs(cos((guess + s * d) * arccos(x)) - Tx) < 1e-10:
                return guess + s * d
[...] using the Lenstra–Lenstra–Lovász (LLL) lattice basis reduction algorithm is something that we didn’t have on our radar screen. It successfully breaks the entire encryption scheme in almost no time.

The attack is reproducible and well-documented. Congratulations for this great work!

— ‘RVB’ authors’ withdrawal notice
RaCoSS

- ‘Random Code-based Signature Scheme’.
- Attack joint w/ Bernstein, Hülsing, Lange.
RaCoSS

- ‘Random Code-based Signature Scheme’.
- Attacks joint w/ Bernstein, Hülsing, Lange.
RaCoSS

- \( n = 2400, k = 2060. \)
- Public parameter: fixed random matrix \( H \in \mathbb{F}_2^{(n-k)\times n}. \)
- **Secret key** is sparse \( S \in \mathbb{F}_2^{n\times n}. \) **Public key** is \( T = H \cdot S. \)
RaCoSS

- $n = 2400, k = 2060$.
- Public parameter: fixed random matrix $H \in \mathbb{F}_2^{(n-k) \times n}$.
- **Secret key** is sparse $S \in \mathbb{F}_2^{n \times n}$. **Public key** is $T = H \cdot S$.
- Hash function $\text{wrhf}$ maps to $n$-bit strings of weight 3.
- **Signing** a message $m$: Pick a low-weight $y \in \mathbb{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output $(z, c)$. 
\begin{itemize}
  \item $n = 2400, \; k = 2060$.
  \item Public parameter: fixed random matrix $H \in \mathbb{F}_2^{(n-k) \times n}$.
  \item **Secret key** is sparse $S \in \mathbb{F}_2^{n \times n}$. **Public key** is $T = H \cdot S$.
  \item Hash function $\text{wrhf}$ maps to $n$-bit strings of weight 3.
  \item **Signing** a message $m$: Pick a low-weight $y \in \mathbb{F}_2^n$.
    Compute $v = Hy, \; c = h(v, m), \; z = Sc + y$. Output $(z, c)$.
  \item **Verifying** $m, (z, c)$: Check that weight$(z) \leq 1564$.
    Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.
\end{itemize}
Implementation bug:

```c
unsigned char  c[RACOSS_N];
unsigned char  c2[RACOSS_N];

/* ... */

for( i=0 ; i<(RACOSS_N/8) ; i++ )
    if( c2[i] != c[i] )
        /* fail */

return 0; /* accept */
```
Implementation bug:

unsigned char  c[RACOSS_N];
unsigned char  c2[RACOSS_N];

/* ... */

for( i=0 ; i<(RACOSS_N/8) ; i++ )
  if( c2[i] != c[i] )
    /* fail */

return 0; /* accept */
RaCoSS

Implementation bug:

    unsigned char  c[RACOSS_N];
    unsigned char  c2[RACOSS_N];

    /* ... */

    for( i=0 ; i<(RACOSS_N/8) ; i++ )
        if( c2[i] != c[i] )
            /* fail */

    return 0; /* accept */

...compares only the first 300 coefficients!
Thus, a signature with $c[0...299] = 0$ is accepted for

$$\binom{2100}{3}/\binom{2400}{3} \approx 67\%$$

of all messages.
The weight-restricted hash function (wrhf):

- maps to 2400-bit strings of weight 3.
RaCoSS

The weight-restricted hash function (wrhf):

- maps to 2400-bit strings of weight 3.
- only $\binom{2400}{3} \approx 2^{31}$ possible outputs.
The weight-restricted hash function (wrhf):

- maps to 2400-bit strings of weight 3.
- only $\binom{2400}{3} \approx 2^{31}$ possible outputs.
- slow: 600 to 800 hashes per second and core.
- expected time for a preimage on $\approx 100$ cores: 10 hours.
- crashed while brute-forcing: memory leaks 😞
The weight-restricted hash function (wrhf):

- maps to 2400-bit strings of weight 3.
- only $\binom{2400}{3} \approx 2^{31}$ possible outputs.
- slow: 600 to 800 hashes per second and core.
- expected time for a preimage on $\approx 100$ cores: 10 hours.
- crashed while brute-forcing: memory leaks 🙁
- another message signed by the first KAT:

```
NISTPQC is so much fun! 10900qmmP
```
RaCoSS ✓

Linear algebra!

- Computing $z$ is the only step of signing that uses $S$.
- Problem: Find ‘low-weight’ $z$ such that $v = Hz + Tc$. 

---
Linear algebra!

- Computing $z$ is the only step of signing that uses $S$.
- Problem: Find ‘low-weight’ $z$ such that $v = Hz + Tc$.
- Pick $n - k$ columns of $H$ that form an invertible matrix $H_1$.\(^2\)
  This can be written as $H_1 = HC$ with $C \in \mathbb{F}_2^{n \times (n-k)}$.\(^3\)

\(^2\)Approximately 29% of all $(n - k) \times (n - k)$ matrices over $\mathbb{F}_2$ are invertible.
\(^3\)For the proposed $H$, the first $n - k$ columns of $H$ work: $C = (I_{n-k} \mid 0)^T$. 

\[\text{RaCoSS } \checkmark\]
Linear algebra!

- **Computing** $z$ is the only step of signing that uses $S$.
- **Problem:** Find ‘low-weight’ $z$ such that $v = Hz + Tc$.
- **Pick** $n - k$ columns of $H$ that form an invertible matrix $H_1$.\(^2\)

This can be written as $H_1 = HC$ with $C \in \mathbb{F}_2^{n \times (n-k)}$.\(^3\)

- **Compute** $z_1 := H_1^{-1}(v + Tc)$ and $z := Cz_1$.
- **Then** $Hz = HCz_1 = H_1z_1 = v + Tc$ as desired.

\(^2\) Approximately 29% of all $(n - k) \times (n - k)$ matrices over $\mathbb{F}_2$ are invertible.

\(^3\) For the proposed $H$, the first $n - k$ columns of $H$ work: $C = (I_{n-k} | 0)^T$. 

---

RaCoSS ✓
Linear algebra!

- **Computing** $z$ is the only step of signing that uses $S$.
- Problem: Find ‘low-weight’ $z$ such that $v = Hz + Tc$.
- Pick $n - k$ columns of $H$ that form an invertible matrix $H_1$. This can be written as $H_1 = HC$ with $C \in \mathbb{F}_2^{n \times (n-k)}$.
- Compute $z_1 := H_1^{-1}(v + Tc)$ and $z := Cz_1$.
- Then $Hz = HCz_1 = H_1z_1 = v + Tc$ as desired.
- Expected **weight** of $z$ is $\approx \frac{n-k}{2} = 170 \ll 1564$.
- Properly generated signatures have weight$(z) \approx 261$.

---

2 Approximately 29% of all $(n-k) \times (n-k)$ matrices over $\mathbb{F}_2$ are invertible.
3 For the proposed $H$, the first $n - k$ columns of $H$ work: $C = (I_{n-k} | 0)^T$. 
Cautionary notice

These stunts were performed by trained professionals.
Cautionary notice

These stunts were performed by trained professionals.

You should totally try this at home!
Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can’t break. It’s not even hard. What is hard is creating an algorithm that no one else can break, even after years of analysis.

— Bruce Schneier
Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can’t break. It’s not even hard. What is hard is creating an algorithm that no one else can break, even after years of analysis.

— Bruce Schneier

Thanks!