Implementation of Lattice-Based Signature Scheme Ring-Tesla and Comparison with ECDSA

Kenneth Xu
Stanford University

Abstract
In search of efficient post-quantum alternative signature schemes, lattice-based schemes like BLISS and GLP have become promising fields of research. In this paper we provide an open-source implementation of the Ring-TESLA scheme [1], which is based on the TESLA signature scheme by Alkim et al. [2]. Ring-TESLA is not only theoretically as efficient as the BLISS and GLIP schemes, but also has provably secure instantiation.

We compare the speed of our Ring-Tesla implementation with an ECDSA implementation for two different security levels.

1 Introduction
Our implementation closely follows the description provided by Akleylet et al. 2016. Ring-TESLA has a security reduction from the R-LWE problem [3]. As long as R-LWE is computationally hard, Ring-Tesla is unforgeable against the chosen-message attack. [1]

1.1 Advantages over BLISS and GLP
Ring-Tesla has a stronger security argument since it achieves both good performance with provably secure instantiation, while BLISS and GLP can only achieve one or the other. Provably secure instantiation means parameters are chosen according to the security reduction [1]. Moreover, Ring-Tesla uses uniform-sampling during signature generation, unlike BLISS, which uses Gaussian-sampling, generally assumed to be vulnerable to timing attacks. Comparing the resilience of BLISS, GLP, and Ring-TESLA to fault attacks in Bindel et al. 2016 [4] found that the Ring-TESLA scheme was sensitive to a strict subset of fault attacks affecting the BLISS and GLP.

2 Ring-Tesla Signature Scheme
Ring-Tesla is parameterized by a number of integers: \( n, \omega, d, B, q, U, L, \kappa \) and the security parameter \( \lambda \) where \( n > \kappa > \lambda \). \( n \) is a positive power of 2 and \( q \) is a prime where \( q = 1 \mod 2n \).

The quotient ring of polynomials we work with is defined as \( R_q = \mathbb{Z}_q[x]/(x^n + 1) \) - in other words, all polynomials of degree up to \( n - 1 \) with coefficients in the range \((-\frac{q}{2}, \frac{q}{2})\). The signature scheme also uses a Gaussian distribution \( D_\sigma \) (with standard deviation \( \sigma \)), a Hash function \( H : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa \), and an encoding function \( F \) which maps the binary output of \( H \) to a vector of length \( n \) and weight \( \omega \). We implemented a similar encoding scheme as found in Gneysu et al. [5].

We provide a mathematical overview of the Ring-Tesla algorithm below:

2.1 Globals
\( a_1 \) and \( a_2 \) are global polynomials uniformly sampled from \( R_q \).

2.2 Key Generation
The pseudocode below is borrowed from Akleylet et al. [1]

\[
\text{KeyGen}(1^\lambda; a_1, a_2):
\]

\[
s, e_1, e_2 \rightarrow D_\sigma^q
\]

If \( \text{checkE}(e_1) = 0 \lor \text{checkE}(e_2) = 0 \):

Restart

\[
t_1 \rightarrow a_1 s + e_1 \mod q
\]

\[
t_2 \rightarrow a_2 s + e_2 \mod q
\]

\[
\text{sk} \rightarrow (s, e_1, e_2)
\]

\[
\text{pk} \rightarrow (t_1, t_2)
\]

return (sk, pk)

We first sample three polynomials \( s, e_1, \) and \( e_2 \) from Gaussian distribution \( D_\sigma \). Each polynomial requires \( n \)}
samples, one for each degree from 0 to \( n - 1 \).
A polynomial passes the checkE function if the sum of
its \( \omega \) largest coefficients is less than \( L \).

2.3 Sign

The pseudocode below is borrowed from Akleylet et al. [1]

\[
\text{Sign}(\mu; a_1, a_2, z, e_1, e_2):
\]

\[
\begin{align*}
 v_1 & \rightarrow a_1 y \pmod{q} \\
 v_2 & \rightarrow a_2 y \pmod{q} \\
 c' & \rightarrow H(\lfloor v_1 \rfloor_d q, \lfloor v_2 \rfloor_d q) \\
 c & \rightarrow F(c') \\
 z & \rightarrow y + ac \\
\end{align*}
\]

# Rejection sampling
\[
\begin{align*}
 w_1 & \rightarrow v_1 - e_1 z \pmod{q} \\
 w_2 & \rightarrow v_2 - e_2 z \pmod{q} \\
\text{If } [w_1]_{2^d}, [w_2]_{2^d} \notin R_{\omega - L}, \forall z \notin R_{B - U}: \\
& \quad \text{Restart} \\
& \quad \text{Return } (z, c') \\
\end{align*}
\]

First, polynomial \( y \) is uniformly sampled from \( R_q \),
with additional constraints on the size of coefficients.
Every coefficient in \( y \) must lie in the range \([-B, B]\) where
\( B \in [0, \frac{q}{2}] \).
We hash the concatenation of the rounded values of \( v_1 \) and \( v_2 \)
and the message \( \mu \) (to sign). This rounding function is defined the following way:
\( \lfloor x \rfloor_d q = \lfloor x \pmod{q} \rfloor_d \) and \( [y]_{2^d} = \lfloor y - [y]_{2^d} \rfloor/2^d \),
where \([y]_{2^d}\) is the mod representation of \( y \) in the range \([-2^{d-1}, 2^{d-1}]\)
and \( /2^d \) defines a quotient group.
The encoding function is applied right after hashing to
produce the signature \((z, c)\).
Before returning, however, we apply rejection sampling
by making sure the coefficients of polynomials \( w_1, w_2, \)
and \( z \) are not too large.

2.4 Verify

The pseudocode below is borrowed from Akleylet et al. [1]
and is similar to the reverse of \text{sign().}

\[
\text{Verify}(\mu; z, c'; a_1, a_2, n, t_1, t_2):
\]

\[
\begin{align*}
 c & \rightarrow F(c') \\
 w'_1 & \rightarrow a_1 z - t_1 c \pmod{q} \\
 w'_2 & \rightarrow a_2 z - t_2 c \pmod{q} \\
 c'' & \rightarrow H(\lfloor w'_1 \rfloor_d q, \lfloor w'_2 \rfloor_d q) \\
\text{If } c' = c'' \land z \in R_{B - U}: \\
& \quad \text{Return } 1 \\
\text{Else: Return } 0
\end{align*}
\]

3 Implementation

We implemented the Ring-Tesla signature scheme in
C++ and compared its speed with that of a ECDSA C++
implementation.

3.1 Selection of Parameters

We selected mostly the same provably secure parameters
as Akleylet et al. [1] described, for two security levels:
80-bit and 128-bit. Like the paper, we will name them
RingTesla-I and RingTesla-II respectively. We changed
the value of \( \omega \) to 16 for easier implementation. Below
are our selected parameters:

<table>
<thead>
<tr>
<th>Parameter set</th>
<th>Security bits</th>
<th>( n )</th>
<th>( \sigma )</th>
<th>( L )</th>
<th>( \omega )</th>
<th>( B )</th>
<th>( U )</th>
<th>( d )</th>
<th>( q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ring-Tesla-I</td>
<td>80</td>
<td>512</td>
<td>30</td>
<td>814</td>
<td>16</td>
<td>2^{93}</td>
<td>21</td>
<td>3996</td>
<td>37</td>
</tr>
<tr>
<td>Ring-Tesla-II</td>
<td>128</td>
<td>512</td>
<td>52</td>
<td>2766</td>
<td>16</td>
<td>2^{37}</td>
<td>25</td>
<td>3998</td>
<td>57</td>
</tr>
</tbody>
</table>

3.2 Code

Our implementation is available at:
https://github.com/kenxu95/rtesla

The code also contains speed an soundness tests (in main.cc).

4 Performance Results

We ran and compared the speed of the three methods
KeyGen(), Sign(), and Verify() on 10,000 random string
messages of 500 characters. We measured the speed
in cpu time. To calculate the speed of each call, we
averaged over all 10,000 trials for each method. Below
are the results (in milliseconds per call):

<table>
<thead>
<tr>
<th>Signature Scheme</th>
<th>KeyGen</th>
<th>Sign</th>
<th>Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ring-Tesla-I</td>
<td>2.61</td>
<td>16.20</td>
<td>4.99</td>
</tr>
<tr>
<td>Ring-Tesla-II</td>
<td>2.49</td>
<td>15.30</td>
<td>4.82</td>
</tr>
<tr>
<td>ECDSA w/ secp160r1</td>
<td>0.92</td>
<td>1.03</td>
<td>1.08</td>
</tr>
<tr>
<td>ECDSA w/ secp192r1</td>
<td>0.67</td>
<td>1.07</td>
<td>1.15</td>
</tr>
<tr>
<td>ECDSA w/ secp224r1</td>
<td>1.71</td>
<td>1.89</td>
<td>2.05</td>
</tr>
<tr>
<td>ECDSA w/ secp256r1</td>
<td>2.81</td>
<td>2.99</td>
<td>3.29</td>
</tr>
<tr>
<td>ECDSA w/ secp256k1</td>
<td>2.14</td>
<td>2.30</td>
<td>2.35</td>
</tr>
</tbody>
</table>

We compared our Ring-Tesla implementation against
five different ECDSA curves, each providing a different
level of bit security. While the trend is for the ECDSA
algorithm to take more time when providing more bits
of security, the same cannot be said for Ring-Tesla.

We are mainly interested in curves secp192r1 and
secp256r1, which provide 80 and 128 bits of security re-
spectively, just like our Ring-Tesla parameter sets.
It is clear that Ring-Tesla is significantly slower when
signing, possibly due to rejection sampling. For 80-bit
security, key generation and verification in ECDSA are a few times faster. However, for 128-bit security, the gap between the two schemes shrinks significantly. It is possible that, with sufficient optimization, Ring-Tesla can match the speed of ECDSA key generation and verification.

References


Notes

¹Gus Gutoski and Chris Peikert discovered a mistake in the tight security reduction from the R-LWE problem to TESLA presented in the referenced paper. The mistake, however, does not yet lead to any attack against TESLA. The non-tight security reduction given by Bai and Galbraith still holds.