(Authenticated) Key Exchanges from (Ring) Learning with Errors

Jintai Ding, University of Cincinnati

Joint work with Xiang Xie, Xiaodong Lin, Jiang Zhang, Zhenfeng Zhang, Michael Snook, Özgür Dagdelen

MELTUP Forum, June 19, 2015
The context of our work - PQC

- Shor’s quantum algorithm
- Post-quantum cryptography
  Develop public key cryptosystems that could resist future quantum computer attacks
The Preparation for the Future

- The first Quantum-Safe-Crypto Workshop
  26 - 27 September, 2013

**ETSI – the European Telecommunications Standards Institute** at SOPHIA ANTIPOLIS, FRANCE

- The second Quantum-Safe-Crypto Workshop
  6 - 2 October, 2014, Ottawa, Canada
  White paper

- The Quantum-Safe-Crypto Workshop at **NIST: National Institute of Standard of Technology**,
  April 7-8, 2015, Washington DC
Fig. 1. Seven stages in the development of quantum information processing. Each advancement requires mastery of the preceding stages, but each also represents a continuing task that must be perfected in parallel with the others. Superconducting qubits are the only solid-state implementation at the third stage, and they now aim at reaching the fourth stage (green arrow). In the domain of atomic physics and quantum optics, the third stage had been previously attained by trapped ions and by Rydberg atoms. No implementation has yet reached the fourth stage, where a logical qubit can be stored, via error correction, for a time substantially longer than the decoherence time of its physical qubit components.
Practical Challenge

- Quantum computing will break many public-key cryptographic algorithms/schemes
  - Key agreement (e.g. DH and MQV)
  - Digital signatures (e.g. RSA and DSA)
  - Encryption (e.g. RSA)

- These algorithms have been used to protect Internet protocols (e.g. IPsec) and applications (e.g. TLS)

- NIST is studying “quantum-safe” replacements

- This talk will focus on practical aspects
  - For security, see Yi-Kai Liu’s talk later today
Post Quantum Needs – Functionality

- Key Exchange – for secure communications
- Signatures – for Authentication
Key Exchange Applications — SSL/TLS

- RSA
- Diffie–Hellman
- Our goal — replacements for post quantum world
Diffie-Hellman Key Exchange

\[ (g^b)^a \rightarrow g^a \]

\[ g^b \rightarrow (g^a)^b \]
Generalizing DH

- DH works because maps $f(x) = x^a$ and $h(x) = x^b$ commute

  \[ f \circ h = h \circ f, \]

- composition
- Nonlinearity
- Many attempts – Braid group etc
Generalizing DH

- When do we have commuting *nonlinear* maps?
  - Powers of $x$ (normal DH)
  - Iterates of a polynomial
  - J. Ritt (1923) – Power polynomials, Chebychev polynomials.
    Elliptic curve
Our basic idea:

- (Ring) LWE approximately commutes—use to build DH generalization

From

\[(s_1 \times a) \times s_2 = s_1 \times (a \times s_2)\]

to

\[(as_1 + e_1)s_2 \approx s_1 as_2 \approx (as_2 + e_2)s_1.\]
Learning with Errors [2006, Regev]

\[
\begin{pmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_m
\end{pmatrix}
= 
\begin{pmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & a_{22} & \cdots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{m1} & a_{m2} & \cdots & a_{mn}
\end{pmatrix}
\begin{pmatrix}
  s_1 \\
  s_2 \\
  \vdots \\
  s_n
\end{pmatrix} + 
\begin{pmatrix}
  e_1 \\
  e_2 \\
  \vdots \\
  e_m
\end{pmatrix}
\]

- Approximate system over \(\mathbb{Z}_q\)
- Hard to find \(\vec{s}\) from \(A, \vec{b}\).
- Hard to tell if \(\vec{s}\) even exists
- Reduction to lattice approximation problems
Definition

Let $n$ be a power of 2, $q \equiv 1 \pmod{2^n}$ prime. Define the ring

$$R_q = \frac{\mathbb{Z}_q[x]}{(x^n + 1)}.$$

- Again, $b = as + e$ hard to find $s$
- Hard to distinguish from uniform $b$
- Approximation problems on ideal lattices
- More efficient than standard LWE
Diffie-Hellman from Ideal Lattices

\[ p_A = a s_A + 2 e_A \]

\[ p_B = a s_B + 2 e_B \]

- Public \( a \in R_q \). Acts like generator \( g \) in DH.

Jintai Ding

AKE from rLWE
Diffie-Hellman from Ideal Lattices

\[ p_A = as_A + 2e_A \]
\[ p_B = as_B + 2e_B \]
\[ k_A = s_A p_B = aS_A S_B + 2S_A e_B \approx k_B = p_A s_B = aS_A S_B + 2S_B e_A \]

- Public \( a \in \mathbb{R}_q \). Acts like generator \( g \) in DH.
- Each side’s key is only approximately equal to the other.
- Difference is even—same low bits.
- No authentication—MitM
Difference 2, both even.
Difference 2, both even.

But wait! If $q = 5$, $\mathbb{Z}_q = \{-2, -1, 0, 1, 2\}$.

4 becomes $-1$, now parities disagree!
Motivation
Lattice-based Key Exchange
The Provably Secure Implementations

Lattice Diffie-Hellman
HMQV
Lattice HMQV

Compensating for Wrap-Around

- \( g = 2S_A e_B - 2S_B e_A \).
- Recall: \( |g^{(j)}| < \frac{q}{8} \)
- Define \( E = \{-\lfloor \frac{q}{4} \rfloor, \ldots, \lceil \frac{q}{4} \rceil \} \). Middle half of \( \mathbb{Z}_q \).
- If \( k_B^{(j)} \in E \), no wrap-around occurs; \( k_A^{(j)} \equiv k_B^{(j)} \).
- If \( k_B^{(j)} \notin E \), then \( k_B^{(j)} + \frac{q-1}{2} \in E \)
- If \( k_B^{(j)} \notin E \), \( k_A^{(j)} + \frac{q-1}{2} \equiv k_B^{(j)} + \frac{q-1}{2} \).
Wrap-around Defeated

Define $w^{(j)}_B = \begin{cases} 
0 & k^{(j)}_B \in E, \\
1 & k^{(j)}_B \notin E.
\end{cases}$ Then $k^{(j)}_B + w^{(j)}_B \frac{q-1}{2} \in E$.

Also, $k^{(j)}_B + w^{(j)}_B \frac{q-1}{2} \equiv k^{(j)}_A + w^{(j)}_B \frac{q-1}{2} \pmod{2}$.

- $k^{(j)}_B + w^{(j)}_B \frac{q-1}{2} \mod q \mod 2 = k^{(j)}_A + w^{(j)}_B \frac{q-1}{2} \mod q \mod 2$.
- Wrap-around correction $w_B = (w^{(0)}_B, w^{(1)}_B, \ldots, w^{(n-1)}_B)$
- $\sigma_B = k_B + w_B \frac{q-1}{2} \mod 2$.
- $\sigma_A = k_A + w_B \frac{q-1}{2} \mod 2$. 
Rounding in picture
Authentication: HMQV – To Resist Man-in-the-middle Attack and Achieve Forward Security

- Static keys $a, b$; tied to each party’s identity.
Authentication: HMQV – To Resist Man-in-the-middle Attack and Achieve Forward Security

- Static keys $a, b$; tied to each party’s identity.
- Ephemeral keys $x, y$: **forward security**.

\[
g^a, g^x \rightarrow \quad g^b, g^y
\]
Authentication: HMQV – To Resist Man-in-the-middle Attack and Achieve Forward Security

- Static keys $a$, $b$; tied to each party’s identity.
- Ephemeral keys $x$, $y$: **forward security**.
- Publicly derivable computations $d$, $e$. 

\[
\begin{align*}
\sigma_A & = (g^y (g^b)^e)^{x+da} \\
\sigma_B & = (g^x (g^a)^d)^{y+eb}
\end{align*}
\]
Authentication: HMQV – To Resist Man-in-the-middle Attack and Achieve Forward Security

- Static keys $a, b$; tied to each party’s identity.
- Ephemeral keys $x, y$: **forward security**.
- Publicly derivable computations $d, e$.
- Shared key is $K = H(\sigma_A) = H(\sigma_B)$
HMQV from Ideal Lattices

\[ p_A = a s_A + 2e_A \]
\[ p_B = a s_B + 2e_B \]

- \( p_A, p_B \) as above. Public, static keys for authentication
Motivation
Lattice-based Key Exchange
The Provable Security
Implementations

Lattice Diffie-Hellman
HMQV
Lattice HMQV

HMQV from Ideal Lattices

\[
\begin{align*}
p_A &= as_A + 2e_A, & x_A &= ar_A + 2f_A \\
p_B &= as_B + 2e_B, & y_B &= ar_B + 2f_B
\end{align*}
\]

- \( p_A, p_B \) as above. Public, static keys for authentication
- \( x_A, y_B \) same form. Forward secrecy.

Jintai Ding
AKE from rLWE
**Motivation**

Lattice-based Key Exchange

The Provable Security

Implementations

**Lattice Diffie-Hellman**

**HMQV**

**Lattice HMQV**

**HMQV from Ideal Lattices**

$$p_A = as_A + 2e_A, x_A = ar_A + 2f_A$$

$$p_B = as_B + 2e_B, y_B = ar_B + 2f_B$$

$$k_A = (p_Bd + y_B)(s_Ac + r_A) + 2dg_A$$

$$\approx (aS_Bd + ar_B)(s_Ac + r_A)$$

$$k_B = (p_Ac + x_A)(s_Bd + r_B) + 2cg_B$$

$$\approx (aS_Ac + ar_A)(s_Bd + r_B)$$

- $p_A, p_B$ as above. Public, static keys for authentication
- $x_A, y_B$ same form. Forward secrecy.
- $c, d$ publicly derivable; $g_A, g_B$ random, small.
Key Derivation

Obtaining shared secret from approximate shared secret:

\[ k_A = (k_A^{(0)}, k_A^{(1)}, \ldots, k_A^{(n-1)}) \]
\[ k_B = (k_B^{(0)}, k_B^{(1)}, \ldots, k_B^{(n-1)}) \]
\[ \tilde{g} = (g^{(0)}, g^{(1)}, \ldots, g^{(n-1)}) \]
\[ k_A - k_B = 2\tilde{g} \]
\[ k_A \equiv k_B \pmod{2} \]
Obtaining shared secret from approximate shared secret:

\[ k_A = (k_A^{(0)}, k_A^{(1)}, \ldots, k_A^{(n-1)}) \]
\[ k_B = (k_B^{(0)}, k_B^{(1)}, \ldots, k_B^{(n-1)}) \]
\[ \tilde{g} = (g^{(0)}, g^{(1)}, \ldots, g^{(n-1)}) \]

\[ k_A - k_B = 2\tilde{g} \]
\[ k_A \equiv k_B \pmod{2} \]

- Each \( k_A^{(j)} = k_B^{(j)} + 2g^{(j)} \).
- Each \( g^{(j)} \) is small (\( |g^{(j)}| < \frac{q}{8} \)).
- Matching coefficients differ by small multiple of 2
- Take each coefficient mod 2, get \( n \) bit secret
HMQV from Ideal Lattices—Corrected

\[ p_A, x_A \]
HMQV from Ideal Lattices—Corrected

\[ p_A, x_A \rightarrow k_B \]
HMQV from Ideal Lattices—Corrected

\[ p_A, x_A \]

\[ p_B, y_B, w_B \]

\[ k_B \]
Motivation
Lattice-based Key Exchange
The Provable Security
Implementations

Lattice Diffie-Hellman
HMQV
Lattice HMQV

HMQV from Ideal Lattices—Corrected

\[ p_A, x_A \]
\[ p_B, y_B, w_B \]

\[ k_A \]
\[ k_B \]
HMQV from Ideal Lattices—Corrected

\[ p_A, x_A \]

\[ p_B, y_B, w_B \]

\[ k_A \]

\[ \sigma_A = \sigma_B \]

\[ k_B \]
HMQV from Ideal Lattices—Corrected

\[
\begin{align*}
&k_A \\ &\downarrow \\ &p_A, x_A \\ &\rightarrow \\ &\downarrow \\ &\sigma_A = \sigma_B \\ &\downarrow \\ &H \\ &\downarrow \\ &\text{Key} \\
\end{align*}
\]

\[
\begin{align*}
&k_B \\ &\downarrow \\ &p_B, y_B, w_B \\ &\leftarrow \\ &\downarrow \\ &p_A, x_A
\end{align*}
\]
Proof Games

Proof proceeds by series of games:

- Begin with simulated protocol
- Replace one hash output with true random value, back-program random oracle
- Adversary cannot distinguish from previous game
- Eventually, if original protocol can be distinguished from random, rLWE can be broken
- The modification using rejecting sampling
Forward Security

- If static keys compromised, previous session keys remain secure
- Notion captured in proof by giving adversaries ability to corrupt static key
- Use Bellare–Rogaway model restricted to two-pass
Quantum Hardness

- Proof uses Random Oracle Model—quantum implications not fully understood
- Important step to post quantum key exchange
## Implementations Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>$n$</th>
<th>Security (expt.)</th>
<th>$\alpha$</th>
<th>$\gamma$</th>
<th>$\log \frac{\beta}{\alpha}$</th>
<th>$\log q$ (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I*</td>
<td>1024</td>
<td>80 bits</td>
<td>3.397</td>
<td>101.919</td>
<td>8.5</td>
<td>40</td>
</tr>
<tr>
<td>II</td>
<td>2048</td>
<td>80 bits</td>
<td>3.397</td>
<td>161.371</td>
<td>27</td>
<td>78</td>
</tr>
<tr>
<td>III</td>
<td>2048</td>
<td>128 bits</td>
<td>3.397</td>
<td>161.371</td>
<td>19</td>
<td>63</td>
</tr>
<tr>
<td>IV</td>
<td>4096</td>
<td>128 bits</td>
<td>3.397</td>
<td>256.495</td>
<td>50</td>
<td>125</td>
</tr>
<tr>
<td>V</td>
<td>4096</td>
<td>192 bits</td>
<td>3.397</td>
<td>256.495</td>
<td>36</td>
<td>97</td>
</tr>
<tr>
<td>VI</td>
<td>4096</td>
<td>256 bits</td>
<td>3.397</td>
<td>256.495</td>
<td>28</td>
<td>81</td>
</tr>
</tbody>
</table>
### Communication Overheads

<table>
<thead>
<tr>
<th>Choice of Parameters</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pk</td>
</tr>
<tr>
<td>I*</td>
<td>5 KB</td>
</tr>
<tr>
<td>II</td>
<td>19.5 KB</td>
</tr>
<tr>
<td>III</td>
<td>15.75 KB</td>
</tr>
<tr>
<td>IV</td>
<td>62.5 KB</td>
</tr>
<tr>
<td>V</td>
<td>48.5 KB</td>
</tr>
<tr>
<td>VI</td>
<td>40.5 KB</td>
</tr>
</tbody>
</table>

The bound $6\alpha$ with $\text{erfc}(6) \approx 2^{-55}$ is used to estimate the size of secret keys.
## Timings

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Initiation</th>
<th>Response</th>
<th>Finish</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>3.22 ms (0.02 ms)</td>
<td>8.50 ms (4.69 ms)</td>
<td>5.23 ms (4.73 ms)</td>
</tr>
<tr>
<td>II</td>
<td>12.00 ms (0.04 ms)</td>
<td>29.33 ms (14.64 ms)</td>
<td>17.28 ms (14.61 ms)</td>
</tr>
<tr>
<td>III</td>
<td>10.33 ms (0.04 ms)</td>
<td>25.83 ms (13.46 ms)</td>
<td>15.58 ms (13.40 ms)</td>
</tr>
<tr>
<td>IV</td>
<td>83.61 ms (0.08 ms)</td>
<td>156.58 ms (39.86 ms)</td>
<td>73.11 ms (39.73 ms)</td>
</tr>
<tr>
<td>V</td>
<td>61.74 ms (0.08 ms)</td>
<td>117.81 ms (32.58 ms)</td>
<td>55.64 ms (32.20 ms)</td>
</tr>
<tr>
<td>VI</td>
<td>25.42 ms (0.08 ms)</td>
<td>62.31 ms (31.32 ms)</td>
<td>36.80 ms (31.29 ms)</td>
</tr>
</tbody>
</table>

**Table:** Timings of Proof-of-Concept Implementations in ms (The figures in the parentheses indicate the timings with pre-computing. For comparison, by simply using the “speed” command in openssl on the same machine, the timing for dsa1024 signing algorithm is about 0.7 ms, and for dsa2048 is about 2.3 ms).
Summary

- We build a simple AKE based on RLWE.
- They are provably secure.
- We can prove the Forward Security of the AKE.
- Our preliminary implementations are very efficient.
  Our AKE are strong candidates for the post-quantum world.