Post-quantum key exchange for the TLS protocol from the ring learning with errors problem

Douglas Stebila

Queensland University of Technology

Joint work with **Joppe W. Bos** *(NXP Semiconductors)*,
Craig Costello & **Michael Naehrig** *(Microsoft Research)*

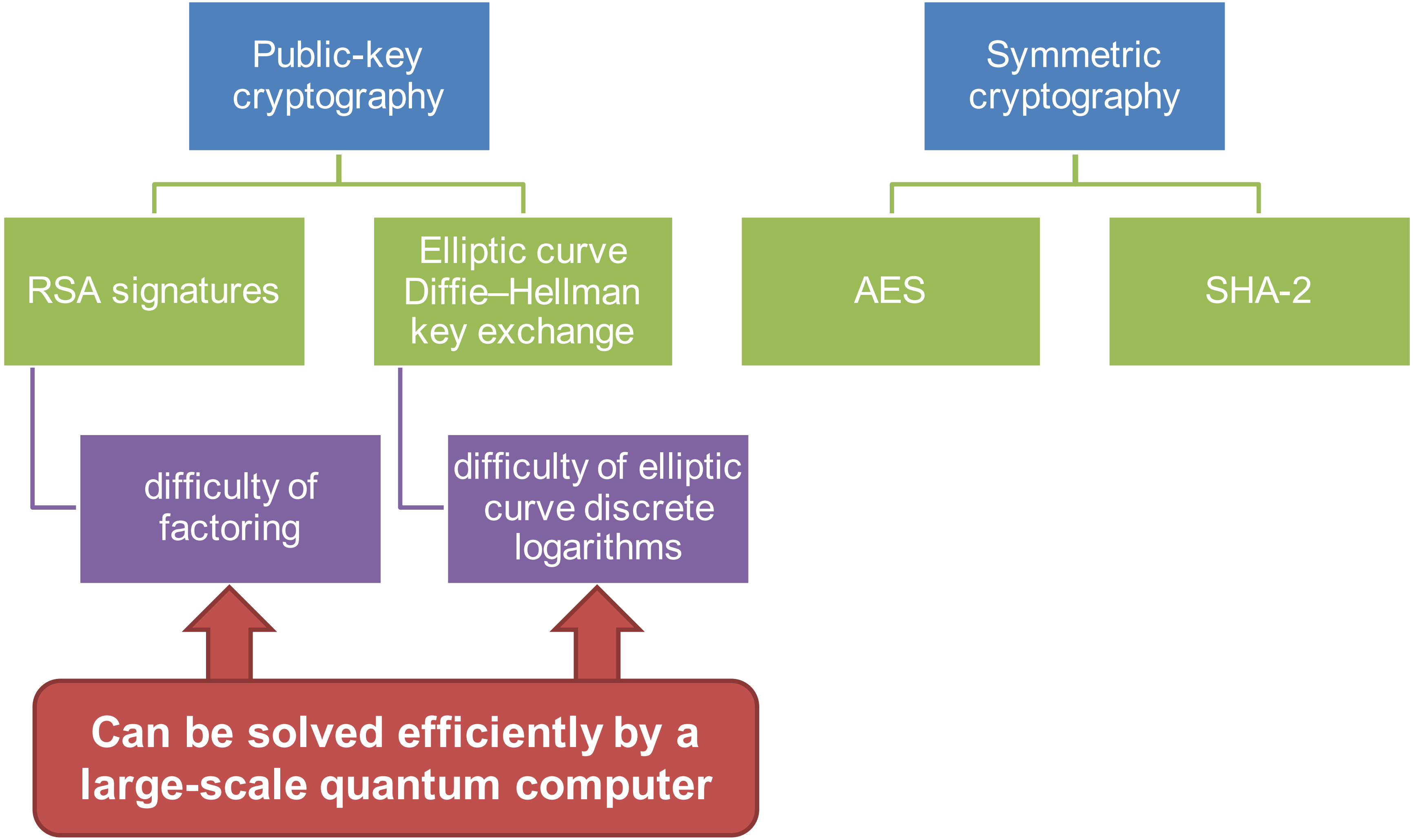
Published at *IEEE Security & Privacy 2015*

CROSSING Collaborative Research Centre, TU Darmstadt • October 1, 2015

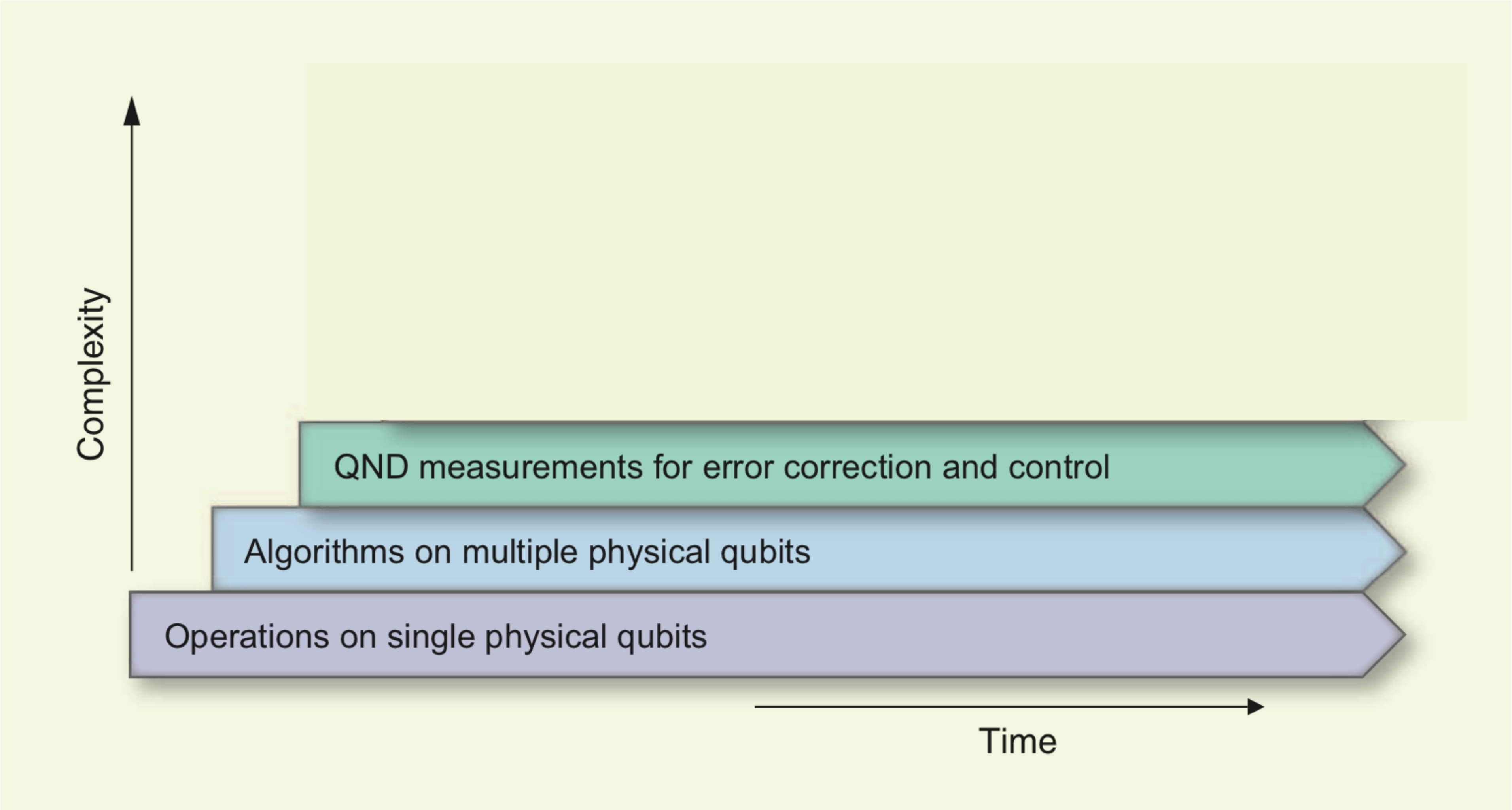
① Motivation

Contemporary cryptography

TLS-ECDHE-RSA-AES128-GCM-SHA256

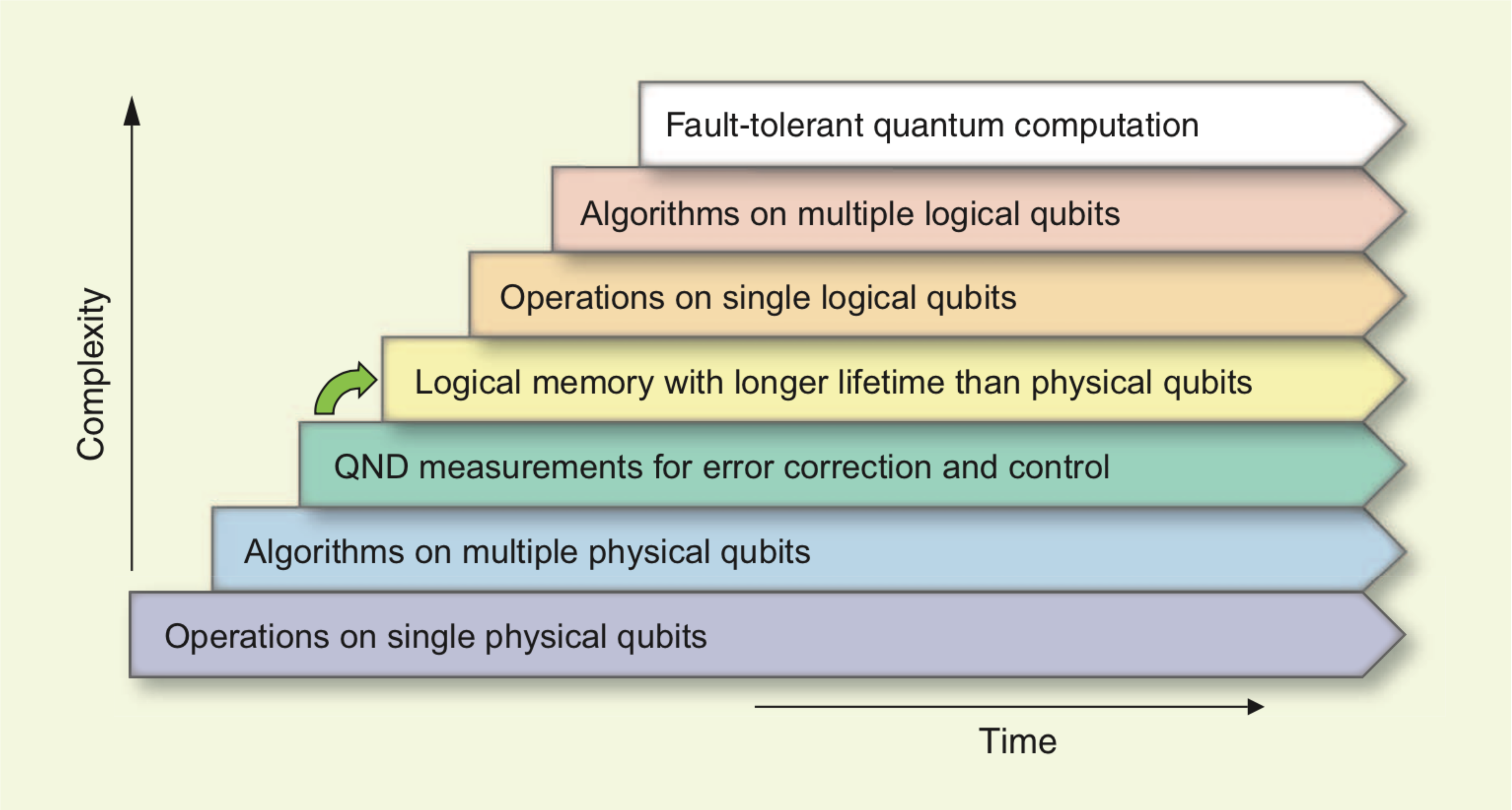


Building quantum computers



Devoret, Schoelkopf. *Science* 339:1169–1174, March 2013.

Building quantum computers



Devoret, Schoelkopf. *Science* 339:1169–1174, March 2013.

Post-quantum / quantum-safe crypto

No known exponential quantum speedup:

Code-based

- McEliece

Hash-based

- Merkle signatures
- Sphincs

Multivariate

- multivariate quadratic

Lattice-based

- NTRU
- learning with errors
- ring-LWE

Lots of questions

- Better classical or quantum attacks on post-quantum schemes?
- What are the right parameter sizes?
- Are the key sizes sufficiently small?
- Can we do the operations sufficiently fast?
- How do we integrate them into the existing infrastructure?

Lots of questions

This talk: ring learning with errors

- Are the key sizes sufficiently small?
- Can we do the operations sufficiently fast?
- How do we integrate them into the existing infrastructure?

This talk: ring-LWE key agreement in TLS

Premise: large-scale quantum computers don't exist right now, but we want to protect today's communications against tomorrow's adversary.

Signatures still done with traditional primitives (RSA/ECDSA)

- we only need authentication to be secure *now*
- benefit: use existing RSA-based PKI

Key agreement done with ring-LWE

② Learning with errors

Solving systems of linear equations

random $\mathbb{Z}_{13}^{7\times 4}$ **secret** $\mathbb{Z}_{13}^{4\times 1}$ $\mathbb{Z}_{13}^{7\times 1}$

$$\begin{pmatrix} 4 & 1 & 11 & 10 \\ 5 & 5 & 9 & 5 \\ 3 & 9 & 0 & 10 \\ 1 & 3 & 3 & 2 \\ 12 & 7 & 3 & 4 \\ 6 & 5 & 11 & 4 \\ 3 & 3 & 5 & 0 \end{pmatrix} \times \begin{pmatrix} \; \\ \; \\ \; \\ \; \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \\ 1 \\ 10 \\ 4 \\ 12 \\ 9 \end{pmatrix}$$

Linear system problem: given **blue**, find **red**

Solving systems of linear equations

random $\mathbb{Z}_{13}^{7\times 4}$					secret $\mathbb{Z}_{13}^{4\times 1}$		$\mathbb{Z}_{13}^{7\times 1}$
4	1	11	10		6		4
5	5	9	5		9		8
3	9	0	10	×	11	=	1
1	3	3	2		11		10
12	7	3	4				4
6	5	11	4				12
3	3	5	0				9

Easily solved using Gaussian elimination (Linear Algebra 101)

Linear system problem: given blue, find red

Learning with errors problem

random $\mathbb{Z}_{13}^{7\times4}$ × **secret** $\mathbb{Z}_{13}^{4\times1}$ + **small noise** $\mathbb{Z}_{13}^{7\times1}$ = $\mathbb{Z}_{13}^{7\times1}$

$$\begin{pmatrix} 4 & 1 & 11 & 10 \\ 5 & 5 & 9 & 5 \\ 3 & 9 & 0 & 10 \\ 1 & 3 & 3 & 2 \\ 12 & 7 & 3 & 4 \\ 6 & 5 & 11 & 4 \\ 3 & 3 & 5 & 0 \end{pmatrix} \times \begin{pmatrix} 6 \\ 9 \\ 11 \\ 11 \end{pmatrix} + \begin{pmatrix} 0 \\ -1 \\ 1 \\ 1 \\ 1 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 4 \\ 7 \\ 2 \\ 11 \\ 5 \\ 12 \\ 8 \end{pmatrix}$$

Learning with errors problem

$$\underset{\substack{\textbf{random}\\ \mathbb{Z}_{13}^{7\times 4}}}{\begin{pmatrix} 4 & 1 & 11 & 10 \\ 5 & 5 & 9 & 5 \\ 3 & 9 & 0 & 10 \\ 1 & 3 & 3 & 2 \\ 12 & 7 & 3 & 4 \\ 6 & 5 & 11 & 4 \\ 3 & 3 & 5 & 0 \end{pmatrix}} \times \underset{\substack{\textbf{secret}\\ \mathbb{Z}_{13}^{4\times 1}}}{\begin{pmatrix} \square \\ \square \\ \square \\ \square \end{pmatrix}} + \underset{\substack{\textbf{small noise}\\ \mathbb{Z}_{13}^{7\times 1}}}{\begin{pmatrix} \square \\ \square \\ \square \\ \square \\ \square \\ \square \\ \square \end{pmatrix}} = \underset{\mathbb{Z}_{13}^{7\times 1}}{\begin{pmatrix} 4 \\ 7 \\ 2 \\ 11 \\ 5 \\ 12 \\ 8 \end{pmatrix}}$$

LWE problem: given **blue**, find **red**

Toy example versus real-world example

$\mathbb{Z}_{13}^{7\times 4}$

4	1	11	10
5	5	9	5
3	9	0	10
1	3	3	2
12	7	3	4
6	5	11	4
3	3	5	0

$\mathbb{Z}_{4093}^{640\times 256}$

256

640

2738	3842	3345	2979	…
2896	595	3607		
377	1575			
2760				
…				

640 × 256 × 12 bits = **245 KiB**

Ring learning with errors problem

random
$\mathbb{Z}_{13}^{7\times 4}$

4	1	11	10
10	4	1	11
11	10	4	1
1	11	10	4
4	1	11	10
10	4	1	11
11	10	4	1

Each row is the cyclic shift of the row above

Ring learning with errors problem

random

$\mathbb{Z}_{13}^{7\times 4}$

4	**1**	**11**	**10**
3	**4**	**1**	**11**
2	**3**	**4**	**1**
12	**2**	**3**	**4**
9	**12**	**2**	**3**
10	**9**	**12**	**2**
11	**10**	**9**	**12**

Each row is the cyclic shift of the row above

…

with a special wrapping rule: *x* wraps to –*x* mod 13.

Ring learning with errors problem

random
$\mathbb{Z}_{13}^{7\times 4}$

4	**1**	**11**	**10**

Each row is the cyclic shift of the row above
…
with a special wrapping rule:
x wraps to –*x* mod 13.

So I only need to tell you the first row.

Ring learning with errors problem

$$\mathbb{Z}_{13}[x]/\langle x^4 + 1\rangle$$

	$4 + 1x + 11x^2 + 10x^3$	random
×	$6 + 9x + 11x^2 + 11x^3$	secret
+	$0 - 1x + 1x^2 + 1x^3$	small noise
=	$10 + 5x + 10x^2 + 7x^3$	

Ring learning with errors problem

$\mathbb{Z}_{13}[x]/\langle x^4 + 1 \rangle$

	$4 + 1x + 11x^2 + 10x^3$	random
×		secret
+		small noise
=	$10 + 5x + 10x^2 + 7x^3$	

Ring-LWE problem: given **blue**, find **red**

Decision ring learning with errors problem

$\mathbb{Z}_{13}[x]/\langle x^4 + 1 \rangle$

	$4 + 1x + 11x^2 + 10x^3$	random
×	$6 + 9x + 11x^2 + 11x^3$	secret
+	$0 - 1x + 1x^2 + 1x^3$	small noise
=	$10 + 5x + 10x^2 + 7x^3$	looks random

Decision ring-LWE problem: given **blue**, distinguish **green** from random

Decision ring learning with errors problem **with small secrets**

$\mathbb{Z}_{13}[x]/\langle x^4 + 1\rangle$

	$4 + 1x + 11x^2 + 10x^3$	random
×	$1 + 0x - 1x^2 + 2x^3$	small secret
+	$0 - 1x + 1x^2 + 1x^3$	small noise
=	$10 + 5x + 10x^2 + 7x^3$	looks random

Decision ring-LWE problem: given **blue**, distinguish **green** from random

Notation

- q: a prime
- n: a power of 2
- $R = \mathbb{Z}[X]/(X^n + 1)$: ring of polynomials in X with integer coefficients, polynomial reduction modulo $X^n + 1$
- $\mathbb{Z}_q$: integers modulo a prime q
- $R_q = \mathbb{Z}_q[X]/(X^n + 1)$: ring of polynomials in X with integer coefficients modulo q, polynomial reduction modulo $X^n + 1$

Decision ring learning with errors problem

Definition. Let n, R, q and R_q be as above. Let χ be a distribution over R, and let $s \xleftarrow{\$} \chi$. Define $O_{\chi,s}$ as the oracle which does the following:

1. Sample $a \xleftarrow{\$} \mathcal{U}(R_q)$, $e \xleftarrow{\$} \chi$,
2. Return $(a, as + e) \in R_q \times R_q$.

The *decision R-LWE problem* for n, q, χ is to distinguish $O_{\chi,s}$ from an oracle that returns uniform random samples from $R_q \times R_q$. In particular, if $\mathcal{A}$ is an algorithm, define the advantage

$$\mathrm{Adv}^{\mathsf{drlwe}}_{n,q,\chi}(\mathcal{A}) = \left| \Pr\left(s \xleftarrow{\$} \chi; \mathcal{A}^{O_{\chi,s}}(\cdot) = 1\right) - \Pr\left(\mathcal{A}^{\mathcal{U}(R_q \times R_q)}(\cdot) = 1\right) \right| .$$

Hardness of DRLWE

Theory:

- Poly-time (quantum) reduction from approximate shortest-independent vector problem (SIVP) on ideal lattices in *R* to DRLWE. [LPR10]

Practice:

- Assume the best way to solve DRLWE is to solve LWE.
- Solving LWE generally involves a lattice reduction problem.
- Albrecht et al. (eprint 2015/046) have hardness estimates.

For 146-bit classical security (≥ 73-bit quantum security), need larger polynomials with larger coefficients.

$n = 1024$, $q = 2^{32}-1$,
chi = discrete Gaussian with parameter sigma = 8/sqrt(2π)

$$\mathbb{Z}_{2^{32}-1}[x]/\langle x^{1024}+1\rangle$$

1024 × 32 bits = **4 KiB**

③ Key agreement

Basic ring-LWE-DH key agreement (unauthenticated)

- Reformulation of Peikert's R-LWE KEM (*PQCrypto 2014*)

public: "big" a in $R_q = \mathbf{Z}_q[x]/(x^n+1)$

Alice

secret:
random "small" s, e in R_q

Bob

secret:
random "small" s', e' in R_q

Alice → Bob: $b = a \cdot s + e$

Bob → Alice: $b' = a \cdot s' + e'$

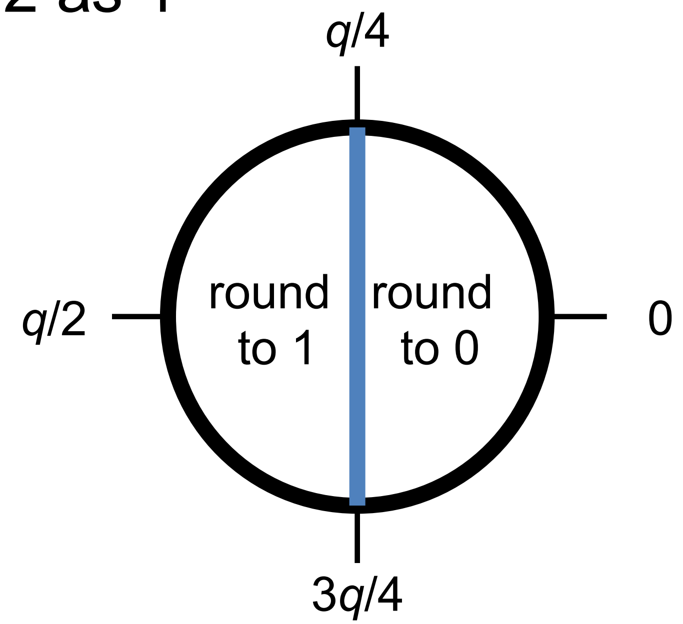
Alice — shared secret:
$s \cdot b' = s \cdot (a \cdot s' + e') \approx s \cdot a \cdot s'$

Bob — shared secret:
$b \cdot s' = (a \cdot s + e) \cdot s' \approx s \cdot a \cdot s'$

These are only approximately equal => need rounding

Basic rounding

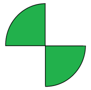
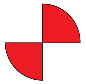
- Each coefficient of the polynomial is an integer modulo q
- Treat each coefficient independently
- Round either to 0 or $q/2$
- Treat $q/2$ as 1

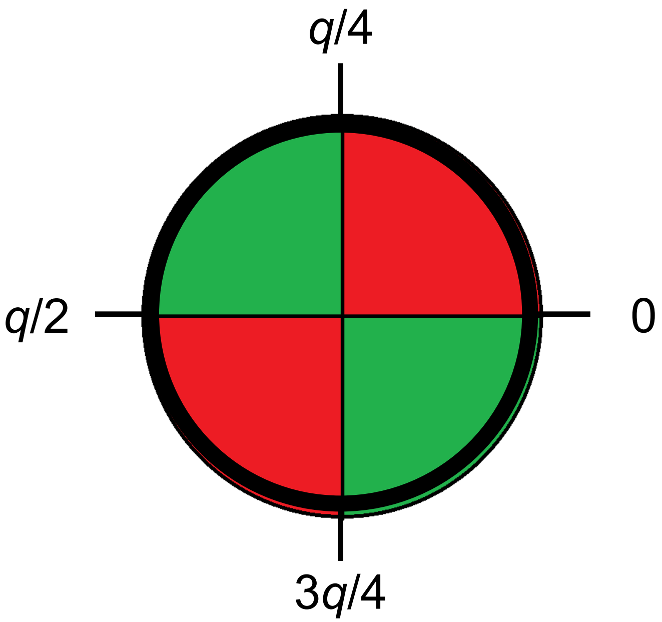


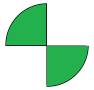
This works most of the time: prob. failure $1/2^{10}$.

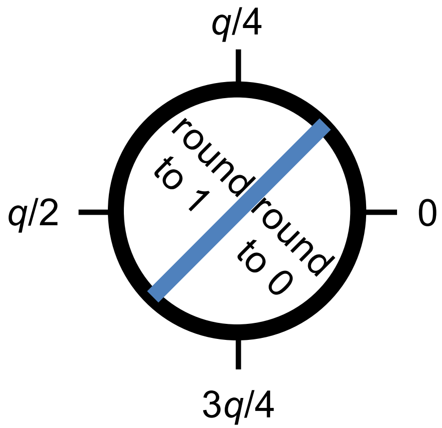
Not good enough: we need exact key agreement.

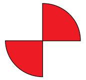
Better rounding (Peikert)

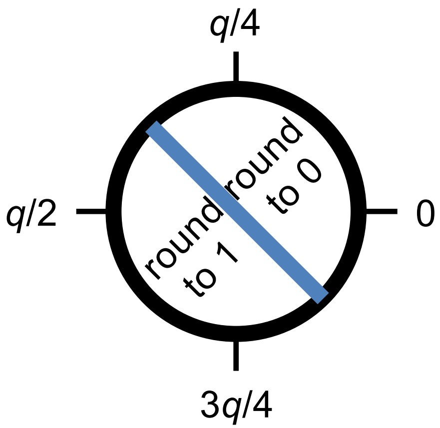
- Bob says which of two regions the value is in:  or 



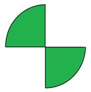
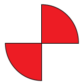
If 



If 



Better rounding (Peikert)

- If $|u–v| \leq q/8$, then this always works.
- For our parameters, probability $|u–v| > q/8$ is less than $2^{-128000}$.
- Security not affected: revealing  or  leaks no information

Exact ring-LWE-DH key agreement (unauthenticated)

- Reformulation of Peikert's R-LWE KEM (*PQCrypto 2014*)

public: "big" a in $R_q = \mathbf{Z}_q[x]/(x^n+1)$

Alice

secret:
random "small" s, e in R_q

Bob

secret:
random "small" s', e' in R_q

Alice → Bob: $b = a \cdot s + e$

Bob → Alice: $b' = a \cdot s' + e'$,  or 

Alice — shared secret:
round($s \cdot b'$)

Bob — shared secret:
round($b \cdot s'$)

Ring-LWE-DH key agreement

Public parameters

Decision R-LWE parameters q, n, χ

$a \xleftarrow{\$} \mathcal{U}(R_q)$

Alice		**Bob**
$s, e \xleftarrow{\$} \chi$		$s', e' \xleftarrow{\$} \chi$
$b \leftarrow as + e \in R_q$	$\xrightarrow{b}$	$b' \leftarrow as' + e' \in R_q$
		$e'' \xleftarrow{\$} \chi$
		$v \leftarrow bs' + e'' \in R_q$
		$\overline{v} \xleftarrow{\$} \mathrm{dbl}(v) \in R_{2q}$
	$\xleftarrow{b', c}$	$c \leftarrow \langle \overline{v} \rangle_{2q,2} \in \{0,1\}^n$
$k_A \leftarrow \mathrm{rec}(2b's, c) \in \{0,1\}^n$		$k_B \leftarrow \lfloor \overline{v} \rceil_{2q,2} \in \{0,1\}^n$

Ring-LWE-DH key agreement

Public parameters

Deci

a

Ali

$s,$

b

Secure if decision ring learning with errors problem is hard.

Decision ring-LWE is hard if a related lattice shortest vector problem is hard.

$$\overline{v} \xleftarrow{\$} \mathrm{dbl}(v) \in R_{2q}$$

$$\xleftarrow{b',c} \qquad c \leftarrow \langle \overline{v} \rangle_{2q,2} \in \{0,1\}^n$$

$$k_A \leftarrow \mathrm{rec}(2b's, c) \in \{0,1\}^n \qquad k_B \leftarrow \lfloor \overline{v} \rceil_{2q,2} \in \{0,1\}^n$$

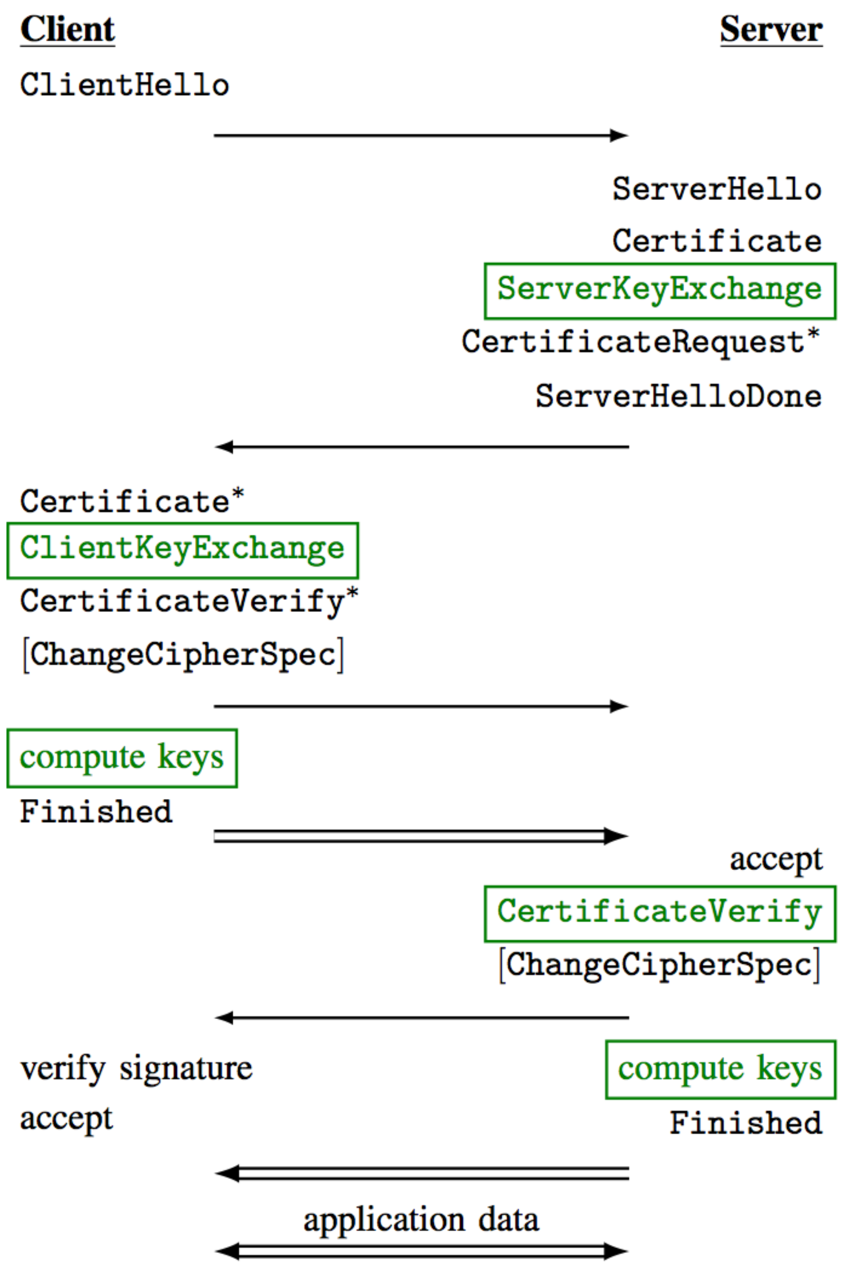
④ Integration into TLS

Integration into TLS 1.2

New ciphersuite:

`TLS-RLWE-SIG-AES128-GCM-SHA256`

- RSA / ECDSA signatures for authentication
- Ring-LWE-DH for key exchange
- AES for authenticated encryption



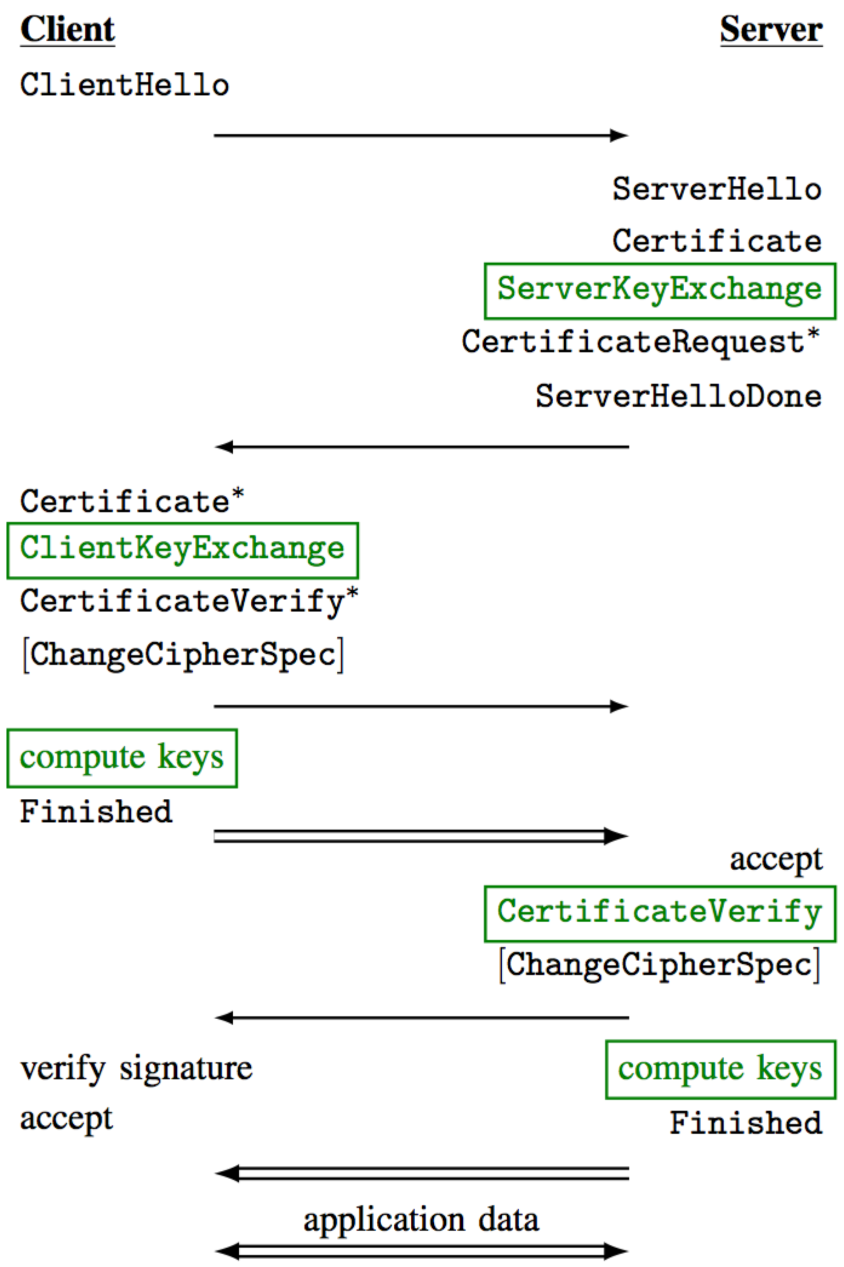
Security within TLS 1.2

Model:

- authenticated and confidential channel establishment (ACCE) (Jager et al., *Crypto 2012*)

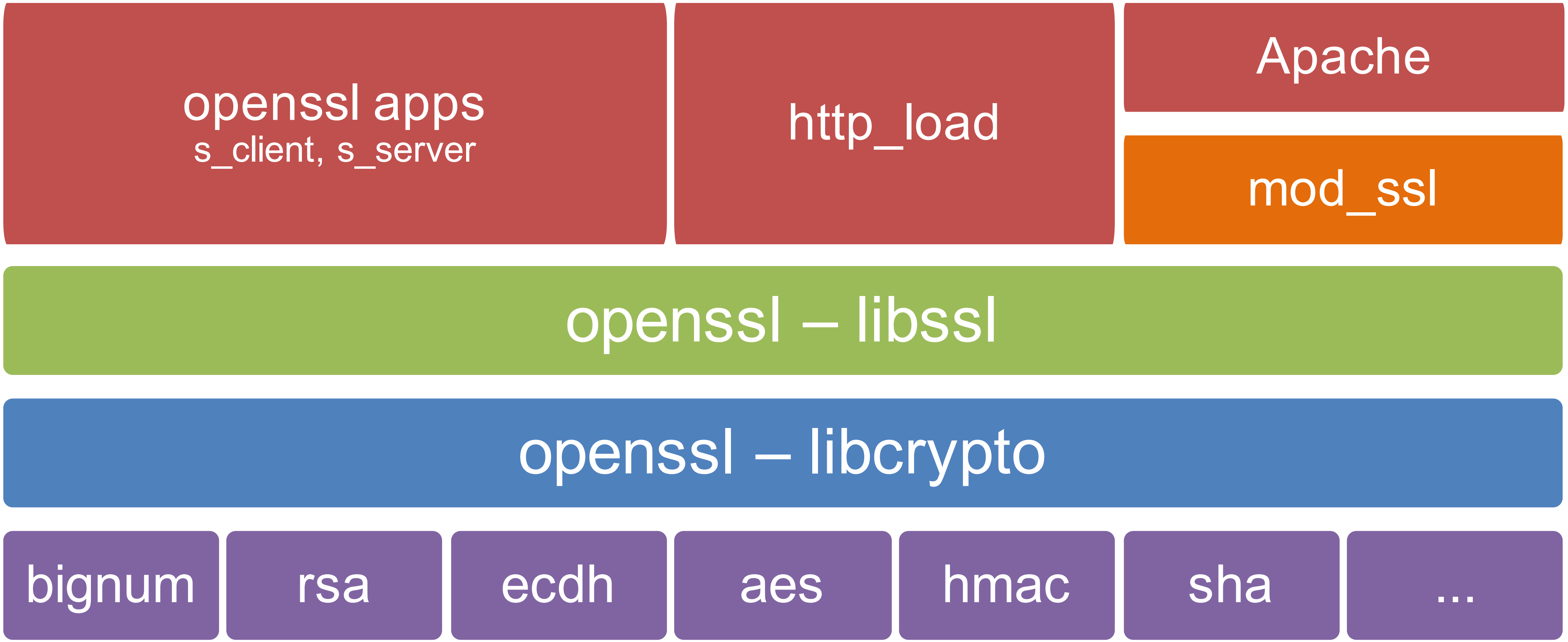
Theorem:

- signed ring-LWE ciphersuite is ACCE-secure if underlying primitives (signatures, ring-LWE, authenticated encryption) are secure
  - Interesting technical detail for ACCE provable security people: need to move server's signature to end of TLS handshake because oracle-DH assumptions don't hold for ring-LWE



⑤ Implementation

OpenSSL stack



Implementation in OpenSSL

No changes needed in openssl apps, http_load, mod_ssl, Apache
(beyond runtime configuration options)

Added ciphersuites in OpenSSL libssl

Wrapped RLWE key exchange into OpenSSL libcrypto

Basic RLWE implemented in standalone C

constant-time

non-constant-time

Implementation aspect 1:
Polynomial arithmetic

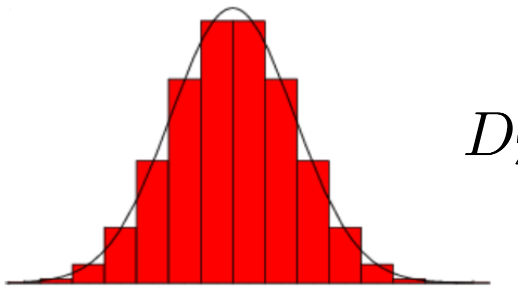
- Polynomial multiplication in $R_q = \mathbf{Z}_q[x]/(x^{1024}+1)$ done with Nussbaumer's FFT:

$$\text{If } 2^m = rk, \text{ then}$$

$$\frac{R[X]}{\langle X^{2^m}+1\rangle} \cong \frac{\left(\frac{R[Z]}{\langle Z^r+1\rangle}\right)[X]}{\langle X^k - Z\rangle}$$

- Rather than working modulo degree-1024 polynomial with coefficients in $\mathbf{Z}_q$, work modulo:
  - degree-256 polynomial whose coefficients are themselves polynomials modulo a degree-4 polynomial,
  - or degree-32 polynomials whose coefficients are polynomials modulo degree-8 polynomials whose coefficients are polynomials
  - or …

Implementation aspect 2:
Sampling discrete Gaussians



$$D_{\mathbb{Z},\sigma}(x) = \frac{1}{S} e^{-\frac{x^2}{2\sigma^2}} \qquad \text{for } x \in \mathbb{Z}, \sigma \approx 3.2, S = 8$$

- Security proofs require "small" elements sampled within statistical distance 2^{-128} of the true discrete Gaussian
- We use inversion sampling: precompute table of cumulative probabilities
  - For us: 52 elements, size = 10000 bits
- Sampling each coefficient requires six 192-bit integer comparisons and there are 1024 coefficients
  - 51 • 1024 for constant time

⑥ Performance testing

Performance – math operations

Operation	Cycles constant-time	Cycles non-constant-time
sample $\stackrel{\$}{\leftarrow} \chi$	1 042 700	668 000
FFT multiplication	342 800	—
FFT addition	1 660	—
$\text{dbl}(\cdot)$ and crossrounding $\langle \cdot \rangle_{2q,2}$	23 500	21 300
rounding $\lfloor \cdot \rceil_{2q,2}$	5 500	3,700
reconciliation $\text{rec}(\cdot, \cdot)$	14 400	6 800

Performance – crypto operations

Operation	Client	Server
R-LWE key generation	0.9ms	0.9ms
R-LWE Alice	0.5ms	
R-LWE Bob		0.1ms
R-LWE total runtime	**1.4ms**	**1.0ms**
ECDH nistp256 (OpenSSL)	**0.8ms**	**0.8ms**

R-LWE 1.75× slower than ECDH

constant-time implementation
Intel Core i5 (4570R), 4 cores @ 2.7 GHz
llvm 5.1 (clang 503.0.30) –O3
OpenSSL 1.0.1f

Performance at different security levels (time in μs)

Operation	80-bit security	146-bit security	453-bit security
s	32	8	5
n	512	1024	2048
sample (constant time)	773	570	665
sample	587	446	652
FFT multiplication	81	168	699
RLWE key generation	1245	1022	1969
RLWE Alice	81	171	740
RLWE Bob	688	602	1392

Disclaimer: these are very preliminary results, and run on a different machine than previous 2 slides so numbers aren't directly comparable.
Programming by Shravan Mishra.

TLS performance testing using Apache/http_load

- Server: Apache web server with mod_ssl and custom OpenSSL
  - Prefork module for multi-threading
  - Disable session resumption
- Client: http_load with custom OpenSSL

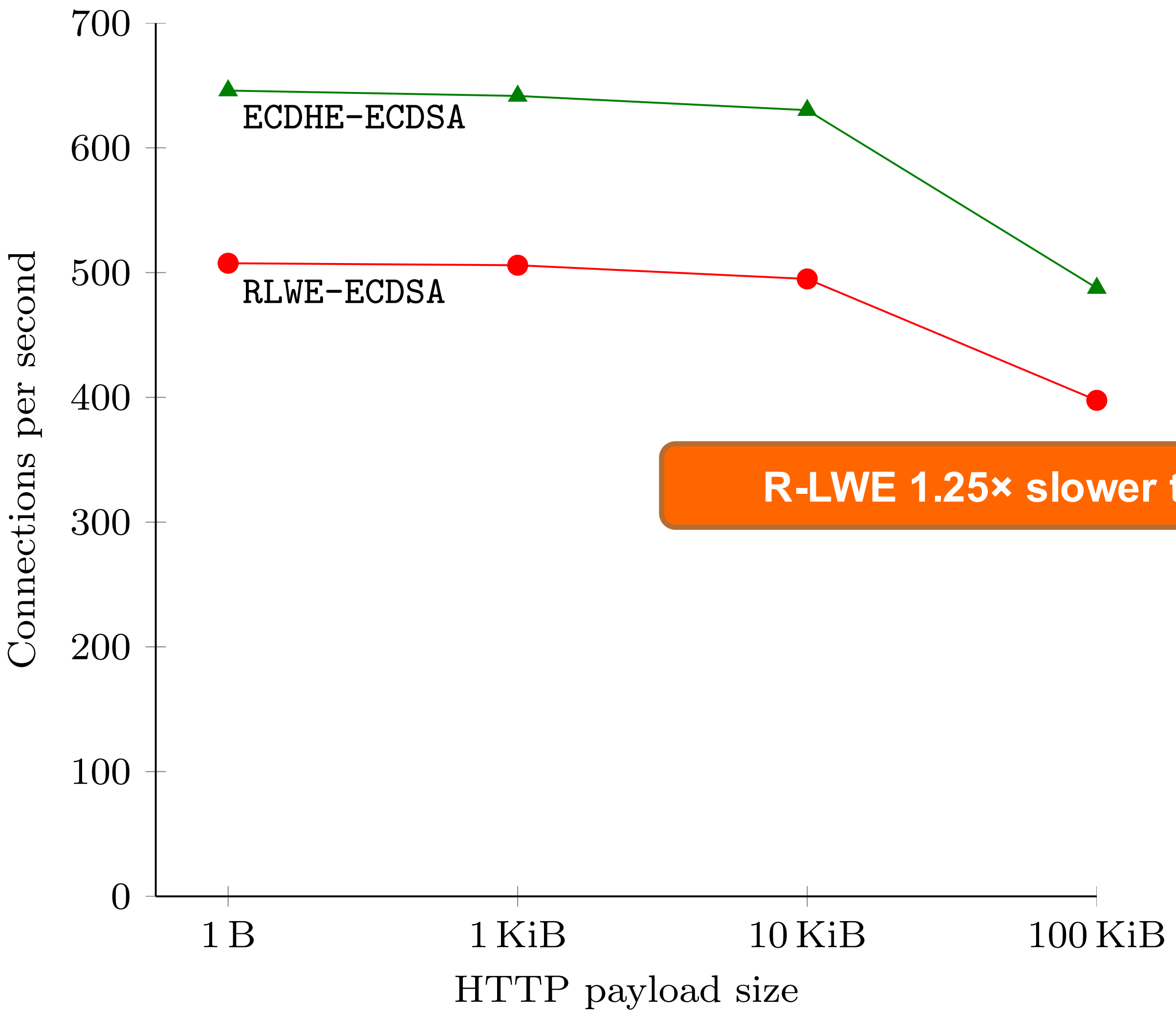
http_load: Multi-threaded HTTP/HTTPS request generator

- Output:
  - mean/min/max time to TCP connect with server
  - mean/min/max time to first HTTP response from server ("latency")
  - total # fetches
    - => avg connections/second

Comments on using http_load

- Client CPU power > server CPU power to ensure clients can fully load server
  - But don’t want to overload server too much or you will get thrashing behaviour
  - Want CPU around 95%
  - http_load tells you latency to first connection, keep increasing number of clients until latency spikes, then back off slightly
- Use multiple parallel executions of http_load
  - i.e., use both multi-threading within http_load and multiple http_load processes
- Should use isolated network
- Run long enough (100s)
- Use multiple runs to collect stdev of fetches across runs

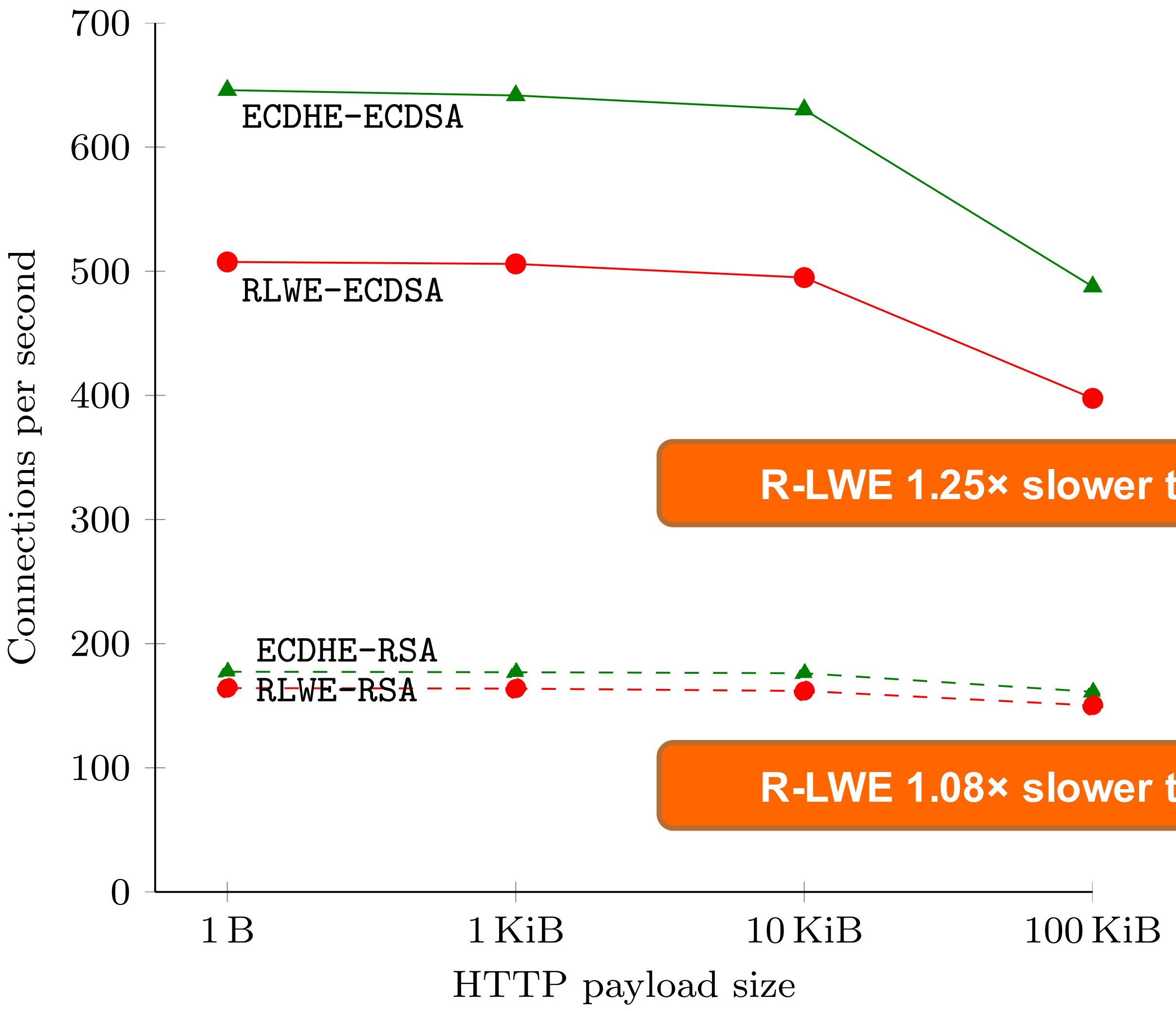
Performance – in TLS



R-LWE 1.25× slower than ECDH

Ring-LWE adds about 8 KiB to handshake size

Performance – in TLS

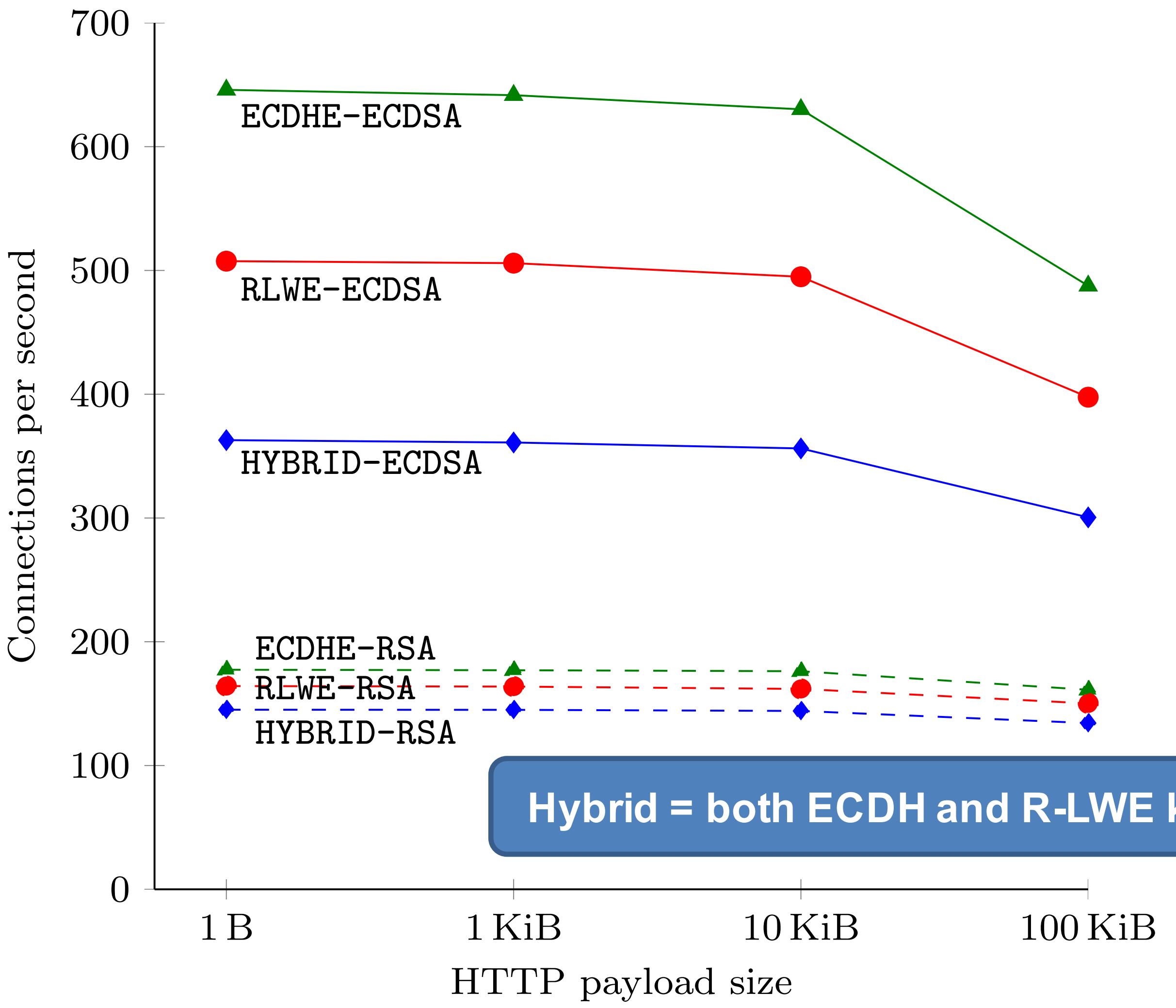


R-LWE 1.25× slower than ECDH

R-LWE 1.08× slower than ECDH

Ring-LWE adds about 8 KiB to handshake size

Performance – in TLS



Hybrid = both ECDH and R-LWE key exchange

Ring-LWE adds about 8 KiB to handshake size

⑦ Summary

Summary

Ring-LWE ciphersuite with traditional signatures:

- Key sizes: not too bad (8 KiB overhead)
- Performance: small overhead (1.1–1.25×) within TLS.
- Integration into TLS: requires reordering messages, but otherwise okay.

Caveat: lattice-based assumptions less studied, algorithms solving ring-LWE may improve, security parameter estimation may evolve.

Future work

better attacks / parameter estimation

- taking into account reduction tightness
- estimate based on best quantum algorithm for solving RLWE

ring-LWE performance improvements

- assembly
- alternative FFT
- better sampling, …

other post-quantum key exchange algorithms

- basic DH directly from LWE
- eCK-secure key exchange
- error correcting codes?

post-quantum authentication

Links

Full version

- http://eprint.iacr.org/2014/599

Magma code:

- http://research.microsoft.com/en-US/downloads/6bd592d7-cf8a-4445-b736-1fc39885dc6e/default.aspx

Standalone C implementation

- https://github.com/dstebila/rlwekex

Integration into OpenSSL

- https://github.com/dstebila/openssl-rlwekex