Evaluation of LWE Key Exchange Protocol

August 17, 2015

1 Asymptotics

Server work	$O(n^2\overline{n})$
Client work	$O(n^2\overline{m})$
Client downloads	$O(n\overline{n})$
Client uploads	$O(n\overline{m})$

2 First round of evaluations

2.1 ECDHE (ECDHE-ECDSA-AES128-GCM-SHA256)

Curve	Client/Server keygen (ms)	Client/Server shared (ms)
160 bit ecdh (secp160r1)	0.2	0.2
192 bit ecdh (nistp192)	0.2	0.2
224 bit ecdh (nistp224)	0.3	0.2
256 bit ecdh (nistp256)	0.6	0.3
384 bit ecdh (nistp384)	1.2	0.6
521 bit ecdh (nistp521)	0.2	1.2
163 bit ecdh (nistk163)	0.2	0.2
233 bit ecdh (nistk233)	0.4	0.2
283 bit ecdh (nistk283)	0.6	0.4
409 bit ecdh (nistk409)	1.5	0.6
571 bit ecdh (nistk571)	0.2	1.5
163 bit ecdh (nistb163)	0.2	0.2
233 bit ecdh (nistb233)	0.4	0.2
283 bit ecdh (nistb283)	0.7	0.4
409 bit ecdh (nistb409)	1.6	0.7
571 bit ecdh (nistb571)	1.6	1.6

TSH protocol transcript:

← ClientHello	158B	0x009e
→ ServerHello	66B	0x0042
→ Certificate	408B	0x0198
→ ServerKeyExchange	125B	0x007d
→ ServerHelloDone	4B	0x4
← ClientKeyExchange	70B	0x0046
← ChangeCipherSpec	1B	0x1
← Finished	16B	0x10
→ Session Ticket	170B	0x00aa
→ ChangeCipherSpec	1B	0x1
→ Finished	16B	0x10

2.2 RLWE (RLWE-ECDSA-AES128-GCM-SHA256)

Keygen: sampling $s(')$, $e(')$ and computing $b(') = as(') + e(')$
Client shared: sampling e'', computing $v = s'b + e''$, $c = \langle v \rangle_2$, $k = \lfloor v \rceil_2$
Server shared: computing $k = rec(b's, c)$

RLWE (128 bits security, deriving 1024 bits key)

Client/Server keygen	Client shared	Server shared
0.674ms	0.402ms	0.119ms

TSH protocol transcript:

← ClientHello	158B	0x009e
→ ServerHello	66B	0x0042
→ Certificate	408B	0x0198
→ ServerKeyExchange	4154B = 4KiB	0x103a
→ ServerHelloDone	4B	0x4
← ClientKeyExchange	4232B = 4KiB	0x1088
← ChangeCipherSpec	1B	0x1
← Finished	16B	0x10
→ Session Ticket	170B	0x00aa
→ ChangeCipherSpec	1B	0x1
→ Finished	16B	0x10

2.3 LWE (LWE-ECDSA-AES128-GCM-SHA256)

Server keygen: sampling S, E and computing $B = AS + E$
Client keygen: sampling S', E' and computing $B' = S'A + E'$
Client shared: sampling E'', computing $V = S'B + E''$, $C = \langle V \rangle_2$, $K = \lfloor V \rceil_2$
Server shared: computing $K = rec(B'S, C)$

LWE (128 bits security, deriving $128 = \overline{nm}$ bits key)

Server keygen	Client keygen	Client shared	Server shared
15.9ms	14.9ms	0.147ms	0.119ms

(*) Storing and using the A^T matrix in generating client's key gives 2× time improvement, otherwise Client keygen takes ≈ 29.5ms.

About 20 X slowdown in key generation compared to RLWE, the rest is comparable.

TSH protocol transcript:

← ClientHello	158B	0x009e
→ ServerHello	66B	0x0042
→ Certificate	408B	0x0198
→ ServerKeyExchange	49209B = 48KiB	0xc039
→ ServerHelloDone	4B	0x4
← ClientKeyExchange	49176B = 48KiB	0xc018
← ChangeCipherSpec	1B	0x1
← Finished	16B	0x10
→ Session Ticket	170B	0x00aa
→ ChangeCipherSpec	1B	0x1
→ Finished	16B	0x10

Time in key generation is split between sampling and matrix multiplication as 26% and 74%. So it make sense to optimize on matrix multiplication first.

2.4 Better reconciliation mechanism

See pencil notes, if B bits are extracted from a single element in Z_q, then the probability for the two parties to fail the handshake (to end up with different keys) would be bounded from above by the following expression:

$$\overline{m} \cdot \overline{n} \cdot (4n) \exp\left(-\frac{q}{2^{3+B} \cdot (2n)\sigma^2}\right)$$

Since everything scales proportionally to $\overline{m}$ and $\overline{n}$, we can draw the following estimates (for now we assume $\overline{n} = \overline{m}$):

Bits extracted from one ring element (B)	Probability of failure	$\overline{n} = \overline{m} = \sqrt{\frac{128}{B}}$	Server/Client keygen (ms)	Client upload/-download (KiB)
1	1e-5553	12	16	48
2	1e-2773	8	10	32
3	1e-1384	7	9	28
4	1e-689	6	8	24
5	1e-341	5	6	20
6	1e-167	5	6	20
7	1e-81	5	6	20
8	1e-37	4	5	16
9	1e-15	4	5	16
10	1e-5	4	5	16
11	1e1	4	5	16