COMSM0097: Advanced Cryptology Course Work Questions

Karl Sachs/eh19257/1914169

December 8, 2022

1 Curve25519

1.a Compute 2P

Curve25519 is a Montgomery curves and so it means that we can do x-only arithmetic on it for fast computation. If we have some $Q \in E(F_{P^2})$ and the function $X()$ [theorem 2.1, CURVE25519], then we can represent $X(Q) = \frac{x}{z}$ where $\frac{x}{z}$ is the quotient of Q. Here $X(2Q)$ or $(Q + Q) = \frac{x_2}{z_2}$, where:

$$x_2 = (x^2 - z^2)^2$$

$$z_2 = 4xz(x^2 + 486662xz + z^2)$$

Using this information we can set our P equal to the quotient $\frac{8}{2}$ and then using the below x-only arithmetic

Thus we get our x-only point as:

$$787053842091199171778545300599777292954332107607667026733043318019255716 5563$$

In Curve25519, the y values don't strictly matter as x-only arithmetic but the point for P is:

$$(787053842091199171778545300599777292954332107607667026733043318019255716 5563,$$

$$643305586501011727775598737492050113802707218027236635362266576763943164 9366$$

And the other point, which would represent $-P$ is:

$$787053842091199171778545300599777292954332107607667026733043318019255716 5563,$$

$$514629887536479804340295051294234527886079201525479156661061262363171331 70583$$

1.b Compute the Image of P

If we have our Curve25519 E in Montgomery form defined as $v^2 = u^3 + 486664u^2 + u$ and we have E' define in Edwards form as $x^2 + y^2 = 1 + (121665/121666)x^2y^2$. Then our isomorphism from $E \mapsto E'$ is:

$$(u, v) \mapsto (\sqrt{486664}(x/y), (x - 1)/(x + 1))$$

It is to be noted that in a Montgomery curve the point at ∞ is the identity of the group. In an Edwards curve the identity/neutral element is found at $(0, 1)$ and so we also need to add a map here to make this an isomorphism.

$$(\infty) \mapsto (0, 1)$$

This would preserve the structure of the group.

Again, when computing the image of P on the Edwards Curve we need to be aware of which one of the two points we actually need to use. We will be using the point that was referred to as P in Q1a, this maps to the image:

(3738745496872751162221495909968467318431947795601340061943607849932863620090 8,

3473762677119485862707129550260637235598099539969216921183727520237393889197 0)

The other, unwanted point (or the image of $-P$) would be:

(2050858964993058608957053340465928074231551437680688140029271350462792861904 1,

3473762677119485862707129550260637235598099539969216921183727520237393889197 0)

1.c Compute $2\phi(P)$

To double a point on an Edwards curve we have to use the formula that is defined below:

$$(x, y) + (x, y) = (\frac{2xy}{1 + dx^2y^2}, \frac{y^2 - x^2}{1 - dx^2y^2})$$

Using the above formula we get $2\phi(P)$ as:

(4038559450440569208610411298014970990647859880813840982516927857318091739386 6,

3301208778665568311973720843369100929116176014367117322780991589887932089594 8)

and $2\phi(-P)$:

(1751045011425240562568137952419424402015639352468187219455951343077564742608 3,

3301208778665568311973720843369100929116176014367117322780991589887932089594 8)

2 Pairings

2.a DLP Oracle

If we have an oracle that can solve the DLP in either $\mathbb{G}_1$, $\mathbb{G}_2$ or $\mathbb{G}_3$ means that we can solve the discrete logarithm problem (DLP) in the remaining 2 groups. We do this by reducing the DLP of each group to the DLP of the group which can be solved by the oracle (this reduction is how the MOV reduction/attack is carried out on cryptographic pairings). By definition, a cryptographic pairing has the property of bilinearlity which makes this reduction possible, bilinearity is defined as:

$$\forall a, b \in \mathbb{F}_q^*, P \in \mathbb{G}_1, Q \in \mathbb{G}_2 : e(aP, bQ) = e(P, Q)^{ab}$$

In the case that we have an oracle that solves $\mathbb{G}_1$, then we need some way to reduce the DLP of $\mathbb{G}_2$ and $\mathbb{G}_3$ to the DLP of $\mathbb{G}_1$.

Say we have $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$, then we can reduce a DLP in $\mathbb{G}_3$ to a DLP $\mathbb{G}_1$ by using the bilinearity property:

$$e(P, Q)^a = e(aP, Q)$$

We can reduce a $\mathbb{G}_2$ DLP to $\mathbb{G}_3$, which we can then reduce to $\mathbb{G}_1$.

$$e(P, bQ) = e(P, Q)^b = e(bP, Q)$$

In the case that we have an oracle that solves $\mathbb{G}_2$, then we need to reduce DLPs in $\mathbb{G}_1$ and $\mathbb{G}_3$ to ones in $\mathbb{G}_2$. $\mathbb{G}_3$ can be reduced to $\mathbb{G}_2$ as follows...

$$e(P,Q)^c = e(P,cQ)$$

... and $\mathbb{G}_1$ to $\mathbb{G}_2$:

$$e(aP,Q) = e(P,Q)^a = e(P,aQ)$$

In the case that we have an oracle that solves the DLP for $\mathbb{G}_3$ then we need a reduction from $\mathbb{G}_1$ to $\mathbb{G}_3$ and $\mathbb{G}_2$ to $\mathbb{G}_3$. The former is as follows:

$$e(aP,Q) = e(P,Q)^a$$

and the latter:

$$e(P,bQ) = e(P,Q)^b$$

Note on efficiency Being able to reduce $\mathbb{G}_3$ DLPs into those in $\mathbb{G}_2$ means that the group doesn't have to use square-and-multiple to compute the discrete logs but can instead use a Montgomery ladder or double-and-add which are much faster algorithms..

2.b CSIDH and Mapping e

In order for e to be generalise the use of pairings we could get e to hold the properties of a pairing so that it works sort of like an extended pairing. To do this we would need to have hold the following properties (i) bilinearity and (ii) non-degenarcy. The property of bilinearity is defined as such:

$$e(g_1 *_s s, g_2 *_s) = (g_1 \bullet g_2) *_T e(s,s)$$

In order for the mapping e to hold the above equality and therefore have the property of bilinearity, we would need:

$$e(s,s) = b(s)$$

In order for the non-degeneracy property to hold we need $e(s,s) \neq 1$.

On top of this we also need the inverse of group actions to be hard (or at least strongly assumed to be hard). In particular, given $g_1, y = g_1 * s$ find s for $* \in \{*_S, *_T\}$

CSIDH wouldn't be suitable for implementing the mapping e. The main issue with this is that CSIDH only has/uses one set whereas e requires 2 sets S and T. CSIDH, being a drop in for diffie-hellman, makes use of a cyclic group which contains elements that are Elliptic curves. Each element in this group has some isogeny between the other elements but if we were to implement e with CSIDH, then we we would get a T which is full of Elliptic curves that may have unknown isogenies between those in this set and thse in our original CSIDH group. Finding these isogenies is, itself, a hard problem and is the problem that makes CSIDH secure, therefore trying to find some isogeny between some elements in CSIDH's implemented S and T would be hard to find.

3 CRYSTALS-dilithium

3.a Brief Summary

CRYSTALS-Dilithium is a latticed based digital signing algorithm (DSA) that has it's hardness based on 2 problems - module learning with errors (MLWE) and SelfTargetMSIS (which, in the Random Oracle Model, can MSIS can be reduced to it). Dilithium has shown to be hard to MLWE and SelfTargetMSIS in the quantum random oracle modle (QROM). MLWE can be defined as given matrix A with elements being polynomials sampled over some ring, vectors s_1 and s_2, u is some uniformly randomly generated data, can we distinguish between the following:

$$(A, t = As_1 + s_2) \qquad (A, u)$$

SelftargetMSIS is defined as find the vector y from:

$$H(m||[I|A] \bullet y) = c$$

The key generation algorithm consists of generating some $k \times l$ matrix A which is full of polynomials uniformly randomly picked from $\mathbf{Z}[X]/(X^n + 1)$. CRYSTALS-dilithium also randomly picks 2 secret vectors s_1 and s_2 which are of size l and k respectively. The value t is defined as $t = As_1 + s_2$ and the public key-secret key pair is returned as $((A, t), (A, t, s1, s2))$ respectively. We can see that the distinguishing the public key (A, t) from some uniformly random sampled t would reduce to the MLWE problem.

Signing requires the signer to generate some a masking vector which contains all polynomial elements. The signer then computes Ay and sets some vector w_1 to be the higher bits of the coefficients of the addition between Ay and $2\gamma_2$ (where γ_2 is a parameter to the dilithium itself). The signer also creates a challenge $c = H(M||w_1)$. It is then used to generate z, a possible signature of the message m, with $z = y + cs_1$. Outputting (z, c) as the signature now could be insecure and in order to mitigate this insecurity the signer can use rejection sampling on y until they have some value of y which leads to secure z. We can see how if we how the challenge c can be reduced to SelfTargetMSIS, therefore meaning the vector of polynomials y is hard to pluck from it.

A verifier computes some w_1' which is formed from taking the high bits of the coefficients of $(Az - ct)$ and accepts the signature of all of the coefficients of z that are less than the scheme's parameter $\gamma_1 - \beta$ (where β is also another parameter to dilithum) and if c is also equal to the hash of the message $||w_1'$. Note: $Az - ct = Ay - cs_2$.

3.b Pros and Cons

3.b.1 Pros

In comparison to with SPHINCS+, signing and verification using AVX2 was significantly faster with CRYSTALS-dilithium. Verification with SPHINCS-SHA2-128-simple would take 800,000 cycles on an AVX2 to complete whereas it would take 300,000 for CRYSTALS-dilithium.

In comparison to Falcon, dilithium is easier to implement and can make use of AES computing hardware to speed up matrix generation. Dilithium would ideally use SHAKE instead of AES to expand the public matrix but the modularity of dilithium means that when hardware accelerated SHAKE becomes the norm, it can be easily and quickly changed.

The repetition of the generation the vector y in the signing of a message is independent of the secret vectors and so the variation in time doesn't leak any information about the secret of the private key. Falcon runs in constant time and so it doesn't leak information about the its secrets which is done by running the algorithm in constant time. This, however, was one such factor that makes the complexity of Falcon be more so than CRYSTALS-dilithium.

3.b.2 Cons

Because both the public key and secret key contains a matrix it means that the key pairs for dilithium are large in comparison to the hash-based scheme SPHINCS+ and to classical signing algorithms.

Both LWE and SIS problems haven't been researched for as long as the one-wayness of hash functions and so there could be some algorithms that aid in the solving of these problems that haven't yet been discovered.

4 Zero-Knowledge

4.a Complete and Sound

To prove completeness, we need to prove that $[z]P = Z + [c]A$:

$$\begin{aligned}[z]P &= [r + xc]P \\ &= [r]P + [xc]P \\ &= [r]P + [x][c]P\end{aligned}$$

$$= [r]P + [c]A \qquad \text{(by } \mathcal{L}\text{)}$$
$$= Z + c[A]$$

and prove $[z]Q = T + [c]B$:

$$[z]Q = [r + xc]Q$$
$$= [r]Q + [xc]Q$$
$$= [r]Q + [x][c]Q$$
$$= [r]Q + [c]B \qquad \text{(by } \mathcal{L}\text{)}$$
$$= T + [c]B$$

We also need to prove soundness. Soundness is defined as if we have some prover which does not know what is being proved, then the verifier will only accept the proof with a small probablility.

$$[z]P = [r + x'c]P$$
$$= [r]P + [c][x']P$$
$$= [r]P + [c]A' \qquad \text{(by } \mathcal{L}\text{)}$$

Given $A = [x]P$, then we want $A = A'$ and therefore we need $x = x'$, where x' is our guess value and x is the value that we actually want to prove. The probability of $x = x'$ is $\frac{1}{|\mathbb{G}|}$ which is equal to $\frac{1}{p}$. If p is large enough, then the probability of the verifier accepting is large enough, therefore proving soundness.

$$[z]Q = [r + x'c]Q$$
$$= [r]Q + [c][x']Q$$
$$= [r]Q + [c]B' \qquad \text{(by } \mathcal{L}\text{)}$$

Following the same logic, given, $B = [x]Q$ then we want $B = B'$ and therefore we need $x = x'$, where x' is our guess value and x is the value we want to prove. Again, the probability of $x = x'$ is $\frac{1}{|\mathbb{G}|} = \frac{1}{p}$. If p is large enough, then the probability of the verifier accepting is large enough, therefore proving soundness.

We have proved soundness for both cases and so the protocol is sound.

4.b Special Sound

Special soundness is defined as given 2 transcripts (Z, T, c, z) and (Z, T, c', z'), we can extract the witness.

$$Z = Z$$
$$[r]P = [r']P$$
$$[z - xc]P = [z' - xc']P$$
$$\Rightarrow$$
$$z - xc = z' - xc'$$
$$xc' - xc = z' - z$$
$$x = \frac{z' - z}{c' - c}$$

We can do the same thing for variable T.

$$T = T$$
$$[r]Q = [r']Q$$
$$[z - xc]Q = [z' - xc']Q$$

$$\Rightarrow$$

$$z - xc = z' - xc'$$
$$xc' - xc = z' - z$$
$$x = \frac{z' - z}{c' - c}$$

Therefore this protocol has special soundness.

4.c (Honest-verifier) Zero-Knowledge

Honest-verifier refers to the assumption that the verifier doesn't lie - i.e. they are honest.

To prove that a protocol is zero-knowledge we need to prove that someone cannot distinguish between a real protocol run and a simulation of a protocol run. We can simulate a protocol run by replacing actual values in the transcript with uniformly randomly sampled data, we would need to simulate Z, T and c. Given some x that is uniformly sampled then if we can assume that g^x can be drawn from a uniformly random distribution, this means that both Z and T would be indistinguishable from random data. c is already randomly sampled and would have no difference between the actual protocol and some simulation.

We can create some simulator which generates a z based off the random inputs Z, T and c. Someone could use this data so check whether the simulated proved has or doesn't have the witness but they wouldn't be able to distinguish between if they are using the real transcript or a simulated transcript.

5 Key Exchanges

5.a X3DH and Double Ratchet

5.a.1 Authentication

X3DH - The identity keys of each party provides authentication of themselves as they are used to sign their corresponding prekeys, which are verified whenever the prekey bundle is received by a party. However, this provides mutual authentication and not full authentication as anyone could technically upload their prekey bundle and say that they're bob.

Double Ratchet does not provide any authentication.

5.a.2 Protection against Person-in-the-Middle attacks

X3DH - A Diffie Hellman exchange is used which ensures that the SK cannot be found. Eve could also carry out a replay attack on Alice and Bob. If some MitM, replays Alice's initial message to bob and Alice didn't use a one-time prekey then BOB can generate the same SK that is used in multiple runs - this is mitigated by using one-time prekeys or by having some ratcheting protocol after (double ratchet).

Double Ratchet - A DH exchange is used in the DH ratchet which stops some MitM from getting shared secrets. If a MitM decides to send messages in the wrong order or skip some messages then this would normally cause issues. Double ratchet mitigates any issues that arise from this by storing header data about messages and ordering them correctly, the protocol will also store the message keys of any skipped messages so that they can be decoded later on. It is to be noted that skipped message keys can't be stored indefinitely and there needs to be some cut off with them, else some MitM could force message skips making the receiver stored a large amount of skipped keys (possibly crashing their system).

5.a.3 Perfect Forward Secrecy

X3DH - During the key agreement when Alice requests Bob's prekey bundle from the server, Alice creates an ephemeral key that is used in 1 or optionally 2 Diffie-Hellman key exchanges; being an ephemeral key it is deleted as soon as it is used to generate the SK therefore providing perfect-forward secrecy.

After some time interval, bob must upload a new signed prekey and once then old private key has been used, it must be deleted - this ensures that previous messages cannot be view by some future adversary.

If a secret prekey and and identity key is compromised, then previous SKs can be generated however the repeated replacement of these prekeys means that not all SKs can be recalculated.

The one-time prekeys also provide perfect forward secrecy in a similar way to how ephemeral keys do.

Double Ratchet - Double ratchet uses a KDF chain which, by definition, means that each new key that is generate is indistinguishable from random data and isn't reversable. This means that if an adversary learns that some key, then they cannot reverse the KDF chain and therefore not recovery previous keys.

5.a.4 Post-Compromise Security

X3DH - If a parties identity key and secret prekey are compromised then future shared secret keys could be calculated, however the frequent replacement of the party's secret prekey means that as soon as a new secret prekey is used, the new SK becomes unknown to the adversary.

Double Ratchet - The DH ratchet in the double ratchet ensures that all future messages are secure if the current sending and receiving chain keys become compromised. This occurs as whenever a party sends a message they generate a DH key pair which becomes their current ratchet key and then they send off the public key with the message.

5.b Implicitly Authenticated Key Exchange

Implicit authentication is when authentication is never directly carried out but is implied when both parties reach a shared secret key. In the exchange below, this was done by combining the ephemeral keys with the long term identity keys to compute the shared secret key, if both parties get the same shared secret key then it means that the each party has the corresponding private key to the public key that they sent to the other - i.e. they are who they say they are.

D	basepoint $= 9$	H
(pk_D, sk_D)		(pk_H, sk_H)
$esk_D \xleftarrow{\$} \mathbb{F}_p$		$esk_H \xleftarrow{\$} \mathbb{F}_p$
$epk_D \leftarrow esk_D * 9$		$epk_H \leftarrow esk_H * 9$
	$\underline{(epk_D, pk_D)} \rightarrow$	
	$\underline{\leftarrow (epk_H, pk_H)}$	
$ssk = epk_H * esk_D + pk_H * sk_D$		$ssk = epk_D * esk_H + pk_D * sk_H$

For completeness:

$$epk_H * esk_D + pk_H * sk_D = epk_D * esk_H + pk_D * sk_H$$

$$(esk_H * 9) * esk_D + (9 * sk_H) * sk_D = (esk_D * 9) * esk_H + (sk_D * 9) * sk_H$$

The ephemeral keys are used to ensure perfect forward secrecy. It is important that as soon as the ssk has been generated the ephemeral keys are deleted for that both D and H. If this is kept around then it could be used (along side a identity key) to decrypt old messages. An ephemeral key should

only be used for one messsage only and then it is to be deleted. This is naturally quite an inefficient process as in order to send a message we have to randomly generate a key, compute a shared secret key and then encrypt. A ratchet could be used if this protocol was expected to encrypt a lot of messages in a short amount of time.

Because the protocol uses Curve25519, it means that it's finding the secret key is based on ECDLP which is assumed to be hard, also due to the choice of prime in curve25519 it's resistant to generic attacks on DLP.

Implementation considerations: The secret ephemeral key needs to be sampled from a perfect uniform distribution however most computers only have access to a pseudo random number generator and so we would need to implement this with either a really good pseudo random number generator or from actual random noise. In Sage, we can use the current random state of sage to generate the secret keys. This is done by calling *...current_randstate().python_random().randrange(self.F.order())* instead of calling *...FiniteField.random_element()*. This improves on Sage's pseudo randomness nature but only slightly and doesn't compare to true random.

Although Sage's elliptic curve implementation is itself secure from side-channel attacks, python is not, This is because python's garbage collector will optimise the code and remove operations in memory that aren't needed. An adversary can use this behaviour to gain information about the secret key.