Introducing CIRCL: An Advanced Cryptographic Library

06/20/2019

 Kris Kwiatkowski  Armando Faz-Hernández

As part of Crypto Week 2019, today we are proud to release the source code of a cryptographic library we've been working on: a collection of cryptographic primitives written in Go, called CIRCL. This library includes a set of packages that target cryptographic algorithms for post-quantum (PQ), elliptic curve cryptography, and hash functions for prime groups. Our hope is that it's useful for a broad

audience. Get ready to discover how we made CIRCL unique.

Cryptography in Go

We use Go a lot at Cloudflare. It offers a good balance between ease of use and performance; the learning curve is very light, and after a short time, any programmer can get good at writing fast, lightweight backend services. And thanks to the possibility of implementing performance critical parts in Go assembly, we can try to 'squeeze the machine' and get every bit of performance.

Cloudflare's cryptography team designs and maintains security-critical projects. It's not a secret that security is hard. That's why, we are introducing the Cloudflare Interoperable Reusable Cryptographic Library - CIRCL. There are multiple goals behind CIRCL. First, we want to concentrate our efforts to implement cryptographic primitives in a single place. This makes it easier to ensure that proper engineering processes are followed. Second, Cloudflare is an active member of the Internet community - we are trying to improve and propose standards to help make the Internet a better place.

Cloudflare's mission is to help build a better Internet. For this reason, we want CIRCL helps the cryptographic community to create proof of concepts, like the post-quantum TLS experiments we are doing. Over the years, lots of ideas have been put on the table by cryptographers (for example, homomorphic encryption, multi-party computation, and privacy preserving constructions). Recently, we've seen those concepts picked up and exercised in a variety of contexts. CIRCL's implementations of cryptographic primitives creates a powerful toolbox for developers wishing to use them.

The Go language provides native packages for several well-known cryptographic algorithms, such as key agreement algorithms, hash functions, and digital signatures. There are also packages maintained by the community under *golang.org/x/crypto* that provide a diverse set of algorithms for supporting authenticated encryption, stream ciphers, key derivation functions, and bilinear pairings. CIRCL doesn't try to compete with *golang.org/x/crypto* in any sense. Our goal is to provide a complementary set of implementations that are more aggressively optimized, or may be less commonly used but have a good chance at being very useful in the future.

Unboxing CIRCL

Our cryptography team worked on a fresh proposal to augment the capabilities of Go users with a new set of packages. You can get them by typing:

```
$ go get github.com/cloudflare/circl
```

The contents of CIRCL is split across different categories, summarized in this table:

Category	Algorithms	Description	Applications
Post-Quantum Cryptography	SIDH	Isogeny-based cryptography.	SIDH provides key exchange mechanisms using ephemeral keys.

	SIKE	SIKE is a key encapsulation mechanism (KEM).	Key agreement protocols.
Key Exchange	X25519, X448	RFC-7748 provides new key exchange mechanisms based on Montgomery elliptic curves.	TLS 1.3. Secure Shell.
	FourQ	One of the fastest elliptic curves at 128-bit security level.	Experimental for key agreement and digital signatures.
Digital Signatures	Ed25519	RFC-8032 provides new digital signature algorithms based on twisted Edwards curves.	Digital certificates and authentication methods.
Hash to Elliptic Curve Groups	Several algorithms: Elligator2,	Protocols based on elliptic curves	Useful in protocols such as

	Ristretto, SWU, Icart.	require hash functions that map bit strings to points on an elliptic curve.	Privacy Pass. OPAQUE. PAKE. Verifiable random functions.
Optimization	Curve P-384	Our optimizations reduce the burden when moving from P-256 to P-384.	ECDSA and ECDH using Suite B at top secret level.

SIKE, a Post-Quantum Key Encapsulation Mechanism

To better understand the post-quantum world, we started experimenting with post-quantum key exchange schemes and using them for key agreement in TLS 1.3. CIRCL contains the sidh package, an implementation of Supersingular Isogeny-based Diffie-Hellman (SIDH), as well as CCA2-secure Supersingular Isogeny-based Key Encapsulation (SIKE), which is based on SIDH.

CIRCL makes playing with PQ key agreement very easy. Below is an example of the SIKE interface that can be used to establish a shared secret between two parties for use in symmetric encryption. The example uses a key encapsulation mechanism (KEM). For our example in this scheme, Alice generates a random secret key, and then uses Bob's pre-generated public key to encrypt (encapsulate) it. The resulting ciphertext is sent to Bob. Then, Bob uses his private key

to decrypt (decapsulate) the ciphertext and retrieve the secret key. See more details about SIKE in this Cloudflare blog.

Let's see how to do this with CIRCL:

```
// Bob's key pair
prvB := NewPrivateKey(Fp503, KeyVariantSike)
pubB := NewPublicKey(Fp503, KeyVariantSike)

// Generate private key
prvB.Generate(rand.Reader)
// Generate public key
prvB.GeneratePublicKey(pubB)

var publicKeyBytes = make([]array, pubB.Size())
var privateKeyBytes = make([]array, prvB.Size())

pubB.Export(publicKeyBytes)
prvB.Export(privateKeyBytes)

// Encode public key to JSON
// Save privateKeyBytes on disk
```

Bob uploads the public key to a location accessible by anybody. When Alice wants to establish a shared secret with Bob, she performs encapsulation that results in two parts: a shared secret and the result of the encapsulation, the ciphertext.

```
// Read JSON to bytes

// Alice's key pair
pubB := NewPublicKey(Fp503, KeyVariantSike)
pubB.Import(publicKeyBytes)

var kem := sike.NewSike503(rand.Reader)
kem.Encapsulate(ciphertext, sharedSecret, pubB)

// send ciphertext to Bob
```

Bob now receives ciphertext from Alice and decapsulates the shared

secret:

```
var kem := sike.NewSike503(rand.Reader)
kem.Decapsulate(sharedSecret, prvA, pubA, ciphertext)
```

At this point, both Alice and Bob can derive a symmetric encryption key from the secret generated.

SIKE implementation contains:

- Two different field sizes: Fp503 and Fp751. The choice of the field is a trade-off between performance and security.

- Code optimized for AMD64 and ARM64 architectures, as well as generic Go code. For AMD64, we detect the micro-architecture and if it's recent enough (e.g., it supports ADOX/ADCX and BMI2 instruction sets), we use different multiplication techniques to make an execution even faster.

- Code implemented in constant time, that is, the execution time doesn't depend on secret values.

We also took care of low heap-memory footprint, so that the implementation uses a minimal amount of dynamically allocated memory. In the future, we plan to provide multiple implementations of post-quantum schemes. Currently, our focus is on algorithms useful for key exchange in TLS.

SIDH/SIKE are interesting because the key sizes produced by those algorithms are relatively small (comparing with other PQ schemes). Nevertheless, performance is not all that great yet, so we'll continue

looking. We plan to add lattice-based algorithms, such as NTRU-HRSS and Kyber, to CIRCL. We will also add another more experimental algorithm called cSIDH, which we would like to try in other applications. CIRCL doesn't currently contain any post-quantum signature algorithms, which is also on our to-do list. After our experiment with TLS key exchange completes, we're going to look at post-quantum PKI. But that's a topic for a future blog post, so stay tuned.

Last, we must admit that our code is largely based on the implementation from the NIST submission along with the work of former intern Henry De Valence, and we would like to thank both Henry and the SIKE team for their great work.

Elliptic Curve Cryptography

Elliptic curve cryptography brings short keys sizes and faster evaluation of operations when compared to algorithms based on RSA. Elliptic curves were standardized during the early 2000s, and have recently gained popularity as they are a more efficient way for securing communications.

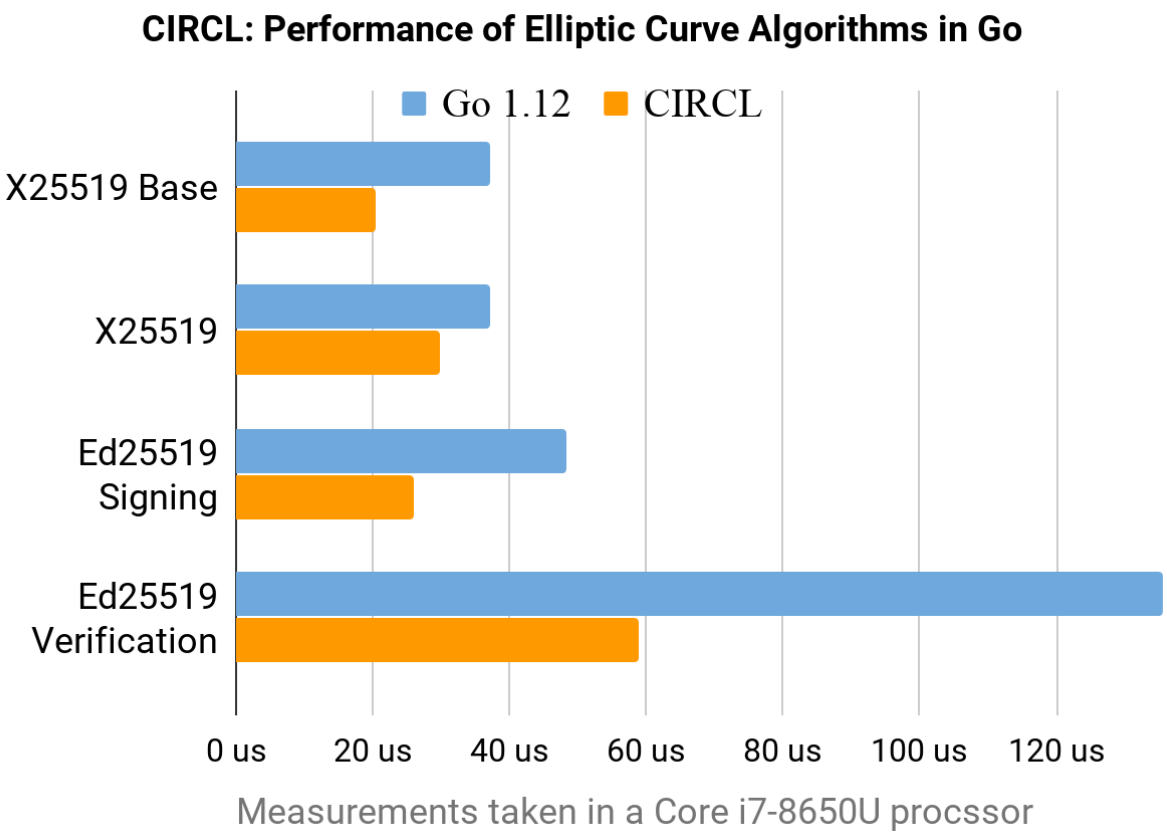
Elliptic curves are used in almost every project at Cloudflare, not only for establishing TLS connections, but also for certificate validation, certificate revocation (OCSP), Privacy Pass, certificate transparency, and AMP Real URL.

The Go language provides native support for NIST-standardized curves, the most popular of which is P-256. In a previous post, Vlad Krasnov described the relevance of optimizing several cryptographic algorithms, including P-256 curve. When working at Cloudflare scale,

little issues around performance are significantly magnified. This is one reason why Cloudflare pushes the boundaries of efficiency.

A similar thing happened on the chained validation of certificates. For some certificates, we observed performance issues when validating a chain of certificates. Our team successfully diagnosed this issue: certificates which had signatures from the P-384 curve, which is the curve that corresponds to the 192-bit security level, were taking up 99% of CPU time! It is common for certificates closer to the root of the chain of trust to rely on stronger security assumptions, for example, using larger elliptic curves. Our first-aid reaction comes in the form of an optimized implementation written by Brendan McMillion that reduced the time of performing elliptic curve operations by a factor of 10. The code for P-384 is also available in CIRCL.

The latest developments in elliptic curve cryptography have caused a shift to use elliptic curve models with faster arithmetic operations. The best example is undoubtedly Curve25519; other examples are the Goldilocks and FourQ curves. CIRCL supports all of these curves, allowing instantiation of Diffie-Hellman exchanges and Edwards digital signatures. Although it slightly overlaps the Go native libraries, CIRCL has architecture-dependent optimizations.



Hashing to Groups

Many cryptographic protocols rely on the hardness of solving the Discrete Logarithm Problem (DLP) in special groups, one of which is the integers reduced modulo a large integer. To guarantee that the DLP is hard to solve, the modulus must be a large prime number. Increasing its size boosts on security, but also makes operations more expensive. A better approach is using elliptic curve groups since they provide faster operations.

In some cryptographic protocols, it is common to use a function with the properties of a cryptographic hash function that maps bit strings into elements of the group. This is easy to accomplish when, for example, the group is the set of integers modulo a large prime. However, it is not so clear how to perform this function using elliptic

curves. In cryptographic literature, several methods have been proposed using the terms *hashing to curves* or *hashing to point* indistinctly.

The main issue is that there is no general method for deterministically finding points on any elliptic curve, the closest available are methods that target special curves and parameters. This is a problem for implementers of cryptographic algorithms, who have a hard time figuring out on a suitable method for hashing to points of an elliptic curve. Compounding that, chances of doing this wrong are high. There are many different methods, elliptic curves, and security considerations to analyze. For example, a vulnerability on WPA3 handshake protocol exploited a non-constant time hashing method resulting in a recovery of keys. Currently, an IETF draft is tracking work in-progress that provides hashing methods unifying requirements with curves and their parameters.

Corresponding to this problem, CIRCL will include implementations of hashing methods for elliptic curves. Our development is accompanying the evolution of the IEFT draft. Therefore, users of CIRCL will have this added value as the methods implement a ready-to-go functionality, covering the needs of some cryptographic protocols.

Update on Bilinear Pairings

Bilinear pairings are sometimes regarded as a tool for cryptanalysis, however pairings can also be used in a constructive way by allowing instantiation of advanced public-key algorithms, for example, identity-based encryption, attribute-based encryption, blind digital signatures, three-party key agreement, among others.

An efficient way to instantiate a bilinear pairing is to use elliptic curves. Note that only a special class of curves can be used, thus so-called *pairing-friendly* curves have specific properties that enable the efficient evaluation of a pairing.

Some families of pairing-friendly curves were introduced by Barreto-Naehrig (BN), Kachisa-Schaefer-Scott (KSS), and Barreto-Lynn-Scott (BLS). BN256 is a BN curve using a 256-bit prime and is one of the fastest options for implementing a bilinear pairing. The Go native library supports this curve in the package golang.org/x/crypto/bn256. In fact, the BN256 curve is used by Cloudflare's Geo Key Manager, which allows distributing encrypted keys around the world. At Cloudflare, high-performance is a must and with this motivation, in 2017, we released an optimized implementation of the BN256 package that is 8x faster than the Go's native package. The success of these optimizations reached several other projects such as the Ethereum protocol and the Randomness Beacon project.

Recent improvements in solving the DLP over extension fields, $GF(p^m)$ for p prime and m>1, impacted the security of pairings, causing recalculation of the parameters used for pairing-friendly curves.

Before these discoveries, the BN256 curve provided a 128-bit security level, but now larger primes are needed to target the same security level. That does not mean that the BN256 curve has been broken, since BN256 gives a security of 100 bits, that is, approximately 2^{100} operations are required to cause a real danger, which is still unfeasible with current computing power.

With our CIRCL announcement, we want to announce our plans for

research and development to obtain efficient curve(s) to become a stronger successor of BN256. According to the estimation by Barbulescu-Duquesne, a BN curve must use primes of at least 456 bits to match a 128-bit security level. However, the impact on the recalculation of parameters brings back to the main scene BLS and KSS curves as efficient alternatives. To this end a standardization effort at IEFT is in progress with the aim of defining parameters and pairing-friendly curves that match different security levels.

Note that regardless of the curve(s) chosen, there is an unavoidable performance downgrade when moving from BN256 to a stronger curve. Actual timings were presented by Aranha, who described the evolution of the race for high-performance pairing implementations. The purpose of our continuous development of CIRCL is to minimize this impact through fast implementations.

Optimizations

Go itself is a very easy to learn and use for system programming and yet makes it possible to use assembly so that you can stay close “to the metal”. We have blogged about improving performance in Go few times in the past (see these posts about encryption, ciphersuites, and image encoding).

When developing CIRCL, we crafted the code to get the best possible performance from the machine. We leverage the capabilities provided by the architecture and the architecture-specific instructions. This means that in some cases we need to get our hands dirty and rewrite parts of the software in Go assembly, which is not easy, but definitely worth the effort when it comes to performance. We focused on x86-64, as this is our main target, but we also think that it’s worth

looking at ARM architecture, and in some cases (like SIDH or P-384), CIRCL has optimized code for this platform.

We also try to ensure that code uses memory efficiently - crafting it in a way that fast allocations on the stack are preferred over expensive heap allocations. In cases where heap allocation is needed, we tried to design the APIs in a way that, they allow pre-allocating memory ahead of time and reuse it for multiple operations.

Security

The CIRCL library is offered as-is, and without a guarantee. Therefore, it is expected that changes in the code, repository, and API occur in the future. We recommend to take caution before using this library in a production application since part of its content is experimental.

As new attacks and vulnerabilities arise over the time, security of software should be treated as a continuous process. In particular, the assessment of cryptographic software is critical, it requires the expertise of several fields, not only computer science. Cryptography engineers must be aware of the latest vulnerabilities and methods of attack in order to defend against them.

The development of CIRCL follows best practices on the secure development. For example, if time execution of the code depends on secret data, the attacker could leverage those irregularities and recover secret keys. In our code, we take care of writing constant-time code and hence prevent timing based attacks.

Developers of cryptographic software must also be aware of optimizations performed by the compiler and/or the processor since

these optimizations can lead to insecure binary codes in some cases. All of these issues could be exploited in real attacks aimed at compromising systems and keys. Therefore, software changes must be tracked down through thorough code reviews. Also static analyzers and automated testing tools play an important role on the security of the software.

Summary

CIRCL is envisioned as an effective tool for experimenting with modern cryptographic algorithms yet providing high-performance implementations. Today is marked as the starting point of a continuous machinery of innovation and retribution to the community in the form of a cryptographic library. There are still several other applications such as homomorphic encryption, multi-party computation, and privacy-preserving protocols that we would like to explore.

We are team of cryptography, security, and software engineers working to improve and augment Cloudflare products. Our team keeps the communication channels open for receiving comments, including improvements, and merging contributions. We welcome opinions and contributions! If you would like to get in contact, you should check out our github repository for CIRCL github.com/cloudflare/circl. We want to share our work and hope it makes someone else's job easier as well.

Finally, special thanks to all the contributors who has either directly or indirectly helped to implement the library - Ko Stoffelen, Brendan McMillion, Henry de Valence, Michael McLoughlin and all the people who invested their time in reviewing our code.

RELATED POSTS

November 01, 2019 1:01PM

La solution Keyless Everywhere

Le temps passe. La vulnérabilité Heartbleed a été découverte il y a un peu plus de cinq ans et demi. Heartbleed a acquis une notoriété mondiale non seulement parce qu'il a été l'un des premiers bugs à posséder sa propre page Web et son...

By Nick Sullivan, Chris Broglie

Crypto Week, Keyless SSL, Seguridad, Crypto, Français

November 01, 2019 1:01PM

Crypto Week 2019 Cryptography Product News Security

Going Keyless Everywhere

Time flies. The Heartbleed vulnerability was discovered just over five and a half years ago. Heartbleed became a household name not only because it was one of the first bugs with its own web page and logo, but because of what it revealed about the fragility of the Internet as a whole....

By Nick Sullivan, Chris Broglie

Crypto Week, Crypto, Security, Keyless SSL

November 01, 2019 1:00PM

Delegated Credentials for TLS

Today we’re happy to announce support for a new cryptographic protocol that helps make it possible to deploy encrypted services in a global network while still maintaining fast performance and tight control of private keys: Delegated Credentials for TLS....

By Nick Sullivan, Watson Ladd

Crypto Week, Crypto, Security, Keyless SSL, Product News

October 31, 2019 1:00PM

Announcing cfnts: Cloudflare's implementation of NTS in Rust

Several months ago we announced that we were providing a new public time service. Part of what we were providing was the first major deployment of the new Network Time Security protocol, with a newly written implementation of NTS in Rust....

By Watson Ladd, Pop Chunhapanya

Crypto Week, Crypto, Security, NTS, Product News