**Hyperledger Cryptography Library Project Proposal HIP HCL v0.3**

**Abstract:**

The goal of this project is to build a shared cryptographic library that would enable people (and projects) to avoid duplicating other cryptographic work and hopefully increase security in the process. The library would be an opt-in repository for projects (and, potentially others) to place and use crypto.

**Sponsors:**

1. Hart Montgomery ([hmontgomery@us.fujitsu.com](mailto:hmontgomery@us.fujitsu.com))
2. Dave Huseby ([dhuseby@linuxfoundation.org](mailto:dhuseby@linuxfoundation.org))
3. Nathan George ([nathan.george@sovrin.org](mailto:nathan.george@sovrin.org))
4. Dan Middleton ([dan.middleton@intel.com](mailto:dan.middleton@intel.com))
5. Mic Bowman ([mic.bowman@intel.com](mailto:mic.bowman@intel.com))
6. Manu Drijvers ([manu@dfinity.org](mailto:manu@dfinity.org))
7. Jan Camenisch ([jan@dfinity.org](mailto:jan@dfinity.org))
8. Binh Nguyen ([bnguyen@statestreet.com](mailto:bnguyen@statestreet.com))
9. Angelo De Caro ([adc@zurich.ibm.com](mailto:adc@zurich.ibm.com))
10. Amit Kumar Gupta ([amit.gupta@sai-infratel.com](mailto:amit.gupta@sai-infratel.com))
11. Vipin Bharathan ([vip@acm.org](mailto:vip@acm.org))
12. Shawn Amundson ([amundson@bitwise.io](mailto:amundson@bitwise.io))

We would like as many sponsors as possible, but we also want people sponsoring this proposal to have some skin in the game. If you’d like to add yourself as a sponsor, please additionally consider adding yourself (and what you can contribute to the project) in the “Effort and Resources” section.

**Dependent Projects:**

The Hyperledger Crypto Library will not depend on any other Hyperledger projects. Instead, we expect and hope that many of the other projects eventually use it as a dependency. This project will have external dependencies in the form of standardized cryptographic libraries (for instance, some of the TLS code).

To be more precise: we expect that Indy will use crypto-lib and depend on it very soon. In fact, this is the reason why we are proposing the project for incubation: The Indy developers do not want to have to depend on something that is still in Hyperledger Labs. In addition, we expect that Fabric will depend on a portion of the library (related to zero knowledge proofs) by the first quarter of next year. We have been in discussions with Sawtooth developers (some of whom are directly involved in crypto-lib) and expect them to use crypto-lib in the future, perhaps after some future major release. The Burrow and Iroha projects have also expressed interest, although we do not have any sort of time frame for adoption (or even certainty of adoption) in their cases.

**Motivation:**

As Hyperledger has matured, the individual projects within Hyperledger have started to find a need for sophisticated cryptographic implementations. Rather than have each project implement its own cryptographic protocols, we think it would be more desirable to collaborate on a shared library. There are many reasons to do this:

1. Avoiding duplication: crypto implementations are notoriously difficult to get correct (particularly when side channels are taken into account) and often require a lot of work in order to complete with a high level of security. The library would potentially allow projects to share crypto implementations, avoiding unnecessary duplication and extra work.
2. Security: having most (or all) of the crypto code in a single location would substantially simplify doing a security analysis of the crypto portion of Hyperledger. In addition, the lack of duplication would mean that maintenance would be easier (and thus, hopefully security bugs would be less numerous). People might also be less likely to “roll their own crypto” if there are easily accessible implementations.
3. Expert Review: In addition, the ability to enforce expert review of all cryptographic code should increase security as well. There has already been at least one substantial bug in a Hyperledger DLT platform at a cryptographic algorithm level. We think that having a concentration of cryptographic experts in Hyperledger will help us minimize the risk of this in the future.
4. Cross-platform interoperability: if two projects use the same crypto libraries, it will simplify (substantially in some cases) cross-platform interoperability, since cryptographic verification will involve the same protocols on both sides.
5. Modularity: This could be the first common component/module and a step towards modular DLT platforms, which share common components. While we have already outlined most of the advantages this modularity would bring in terms of actual functionality, a successful crypto library could encourage and push forward more modular activities.
6. New Projects: It would be easier for new projects to get off the ground if they had easy access to well-implemented, modular cryptographic abstractions.

**Status:**

Currently this is a **proposal**.

**Solution Details:**

We next explain the structure of the project. We aren’t going to explain how cryptography works (which is what this section title might imply) but instead how our project is structured. In a nutshell, this project will consist of various semi-independent *subprojects*. We will have tiered levels of codebases (based on how “safe” we think a particular implementation is) and different types/lists of maintainer (due to the semi-independence of the subprojects and the fact that, as a security-focused project, we need to be especially concerned with code quality). We will explain all of this in detail in this section.

Our project will be segmented in a few different ways. We will have tiers of code, which relate to how basic/standardized the cryptographic algorithms are, as a first segmentation. Furthermore, we will have subprojects, which live in a certain tier but each have independent maintainers. We explain this in more detail below.

Codebase Tiers: We separate cryptographic code into three different “tiers”. We explain these below. The point of this separation is to ensure that users of our library know exactly how safe (or unsafe) the code they are planning on using actually is. The maintainers will initially decide where to put incoming subprojects and be responsible for deciding any “promotions” of codebases.

1. *Standard Implementations*: things that have been through the NIST and/or other standardizations gauntlets and are trusted by the vast majority of the world. Standardized implementations of standard protocols and good interfaces to standardized crypto libraries would go in this library. It is most likely that this tier would consist of subprojects that contain useful “shims” or interfaces to standard libraries. This portion of the library might not be particularly exciting to researchers, but would be very useful to newer or smaller projects (and even to large projects if people develop easy and intuitive interfaces/wrappers). Project members would not be writing new crypto code for this tier of library (at least in their Hyperledger capacity)--all novelty would come in the (hopefully shared) interfaces. Examples of a subproject in this level could be a pluggable API for digital signatures.
2. *Semi-trusted Implementations*: things that people use (and generally trust) but that haven’t been through the standardization gauntlet. For instance, the most commonly used SNARK implementations would fall into this category, as well as perhaps BLS signature implementations (threshold signatures). If we have experts (both in theory and implementation) join the project and decide to add commonly used crypto code (that is stable and, to emphasize again, used in practice) this could be one area where we write new code.
3. *Research Implementations*: this is the tier for cutting-edge cryptography that isn’t widely deployed or well studied. We do not at all encourage people to use cryptography that has not been peer reviewed, so what we intend for this category to include would be something like the following example: someone publishes a much more efficient ZK protocol for something that Hyperledger DLTs are interested in using. The paper is accepted into the CRYPTO conference after the conference review process. Sometime after the crypto acceptance, someone codes up the algorithm. This is code that would qualify as a research implementation.

Tiers 1 and 2 of cryptographic code will be separated within the crypto-lib project. Tier 3 implementations will be placed in Hyperledger Labs and have to go through the usual Hyperledger Labs approval process. We encourage (but certainly cannot require) the Hyperledger Labs stewards to ask the crypto-lib maintainers to review labs projects that are cryptography-based.

Cryptographic Hyperledger labs projects (including those that are research implementations) can be promoted to the crypto-lib project itself in a process described below.

We note that these tiers are very general at this point. In the future, we will want more fine-grained attributes to describe our builds and build recommendations. These will be decided and voted upon (with a two thirds majority vote for changes) by the maintainers.

Subprojects: The Hyperledger Crypto Library will have what we call subprojects. Subprojects are cohesive collections implementations of cryptographic code or interfaces to cryptographic code. Examples of sub-projects might be a BLS threshold signatures sub-project, a pluggable digital signature sub-project, or a ZK-snark implementation sub-project.

Subprojects will make it easier for us to segment our resources and not waste time. We don’t need to involve all of the maintainers of a modular signature implementation in decisions about SNARKs, for instance, and it’s unlikely that all of the people involved in tier 2 projects will want to sit through phone calls or meetings about tier 1 projects. Separating the (highly) different parts of this overall project will make things easier for everyone.

Subprojects will be approved and started by the overwhelming consent of the *crypto-lib maintainers* (see below). While we don’t want to prescribe exactly the criteria for what the maintainers will accept as a subproject, we expect that it will be some combination of cryptographic expertise of subproject maintainers, acceptance in the academic and/or standardization communities of the proposed result, resources committed, and the proposed need of the subproject by the Hyperledger community.

The TSC will have the right to override a subproject approval or rejection.

We comment on the initial proposed subprojects later in the document.

Governance: Given the unique requirements of this project, it is necessary to have a slightly stricter governance structure than currently existing Hyperledger projects. However, the general project governance structure will not diverge from typical open source practices (in fact, we borrow heavily here from the governance models of Fabric and Sawtooth).

The Hyperledger Cryptographic Library will contain *categories* of maintainers which will be represented by separate lists. Each maintainer’s categorization will be based on their contributions and expertise. The goal of having separate maintainer lists is to ensure that all algorithms and code are reviewed by appropriate experts in the area so that we can achieve best-possible security. For instance, we want any code that deviates from standard algorithms or uses cryptographic primitives in a non-black box manner (this means, abstractly, that the “math” is touched) to be reviewed by experts in theoretical cryptography.

Initially our maintainer categories will be simple: we will have one list for each subproject, and one list containing experts in the theory of cryptography (i.e., people who can review algorithms and mathematical modifications to cryptography). In the future, we may add extra lists for more in-depth review as needed (i.e. a post-quantum cryptography expert list or a SNARK expert list).

In general, subproject maintainers will be responsible for setting the direction of the project (or their portion of the project), deciding which features/interfaces to implement, and so forth. Maintainers on the “expert” lists are responsible for ensuring that the project maintains best practices in security and cryptography. We expand on this more below.

*Subproject maintainers* would be maintainers who are responsible for a specific piece (or pieces) of code. For instance, “Lovesh” might be the *maintainer* for an implementation of BLS signatures in a tier 2 repository. “Lovesh” would be responsible (in the traditional maintainer sense) for that particular code. Subproject maintainers could also have responsibility for entire subprojects, but we would leave the decisions about how subprojects appointed maintainers to the sub-projects themselves.

Initial maintainers would be tied to the group contributing code (or a commitment to write code). As a subproject matures, the maintainers may change with overwhelming consensus (a two-thirds majority if a vote is required) from the subproject maintainers.

*Category/expert maintainers* would be maintainers who aren’t (necessarily) attached to any specific pieces of code but are instead experts in a certain domain related to cryptography or security. These maintainerswould be responsible for security oversight of the project as a whole through code and algorithm review of things that are relevant to their expertise. This would include things like making sure currently implemented code meets security requirements, ensuring that new contributions are appropriate, secure, and placed in the proper tier of code (subproject approval), and making sure subprojects don’t take actions that would cause security vulnerabilities.

Our only initial class of category maintainer will be our *theoretical maintainers*, consisting of people who are experienced in the theory of cryptography. We require that at the theoretical maintainers either have an extensive background in academic cryptography and protocol design or in the security issues involved in implementing and using cryptographic protocols. In particular, we need these theoretical maintainers to be able to algorithmically analyze cryptographic protocols, so our ideal maintainer in this category is someone who has reviewed papers (i.e. been on the program committee) for IACR conferences, and anyone who meets this metric and is heavily involved in Hyperledger will probably be asked to be a theoretical maintainer.

The rules for accepting pull requests will start with the typical 2+2 rule, which is being used by Fabric, Sawtooth, and others. However, we will also require that one of the +2s comes from someone on the “theoretical maintainer” list in the case that the coding change uses cryptographic primitives in a non-black box manner (in other words, and informally, touches the math part of the crypto code). This is to ensure that our cryptographic implementations do implement secure algorithms.

*Overall Governance:* the maintainers will make all decisions involving admitting or reclassifying subprojects to the crypto-lib project by general consensus. If there isn’t a good consensus, decisions will come to a vote. We will require a two-thirds vote amongst the maintainers to make an appropriate change. In addition, maintainers can decide (vote) to add more maintainers or remove inactive maintainers.

We expect the well-run subprojects to be almost entirely self-governing. The subproject maintainers will be responsible for setting the direction of their subprojects, choosing new features, and so forth. However, on security-critical issues, given consensus (or a two-thirds vote) from a set of “expert” maintainers, we will allow these “expert” maintainers to mandate subprojects to do make changes related to security flaws. This provision, while unlikely to be used, is needed in case subprojects become neglected or, for some reason, decide to do things that are highly detrimental to security (i.e. roll their own hash functions or make otherwise poor algorithmic choices).

Our initial maintainers will be listed later in this document.

Opt-In Nature:

We want to emphasize that the library would exist only as an opt-in repository. In other words, you wouldn’t have to put your crypto code in there or use crypto code from the library if you didn’t want to do so. This might result in some duplicate implementations in the beginning, but that’s OK for a starting point. Any directives for a Hyperledger project to use code from our library (rather than their own or third party implementations) would have to come from the TSC or maintainers of said other Hyperledger projects rather than this project. We do not want to mandate or intervene in projects--we just want to make things available.

However, as the library matures, we expect that people will find ways to eliminate code duplication and combine resources on a single implementation (perhaps abandoning an implementation, which would be fine as long as it is handled appropriately). In addition, we expect projects (and outside third parties) might want to use the crypto code of other projects, which would be simplified with the use of the library. In the long run, there would hopefully be a minimal set of implementations for each desired crypto functionality.

Building and Using crypto-lib:

How people use each subproject in the crypto-lib will be determined by the subprojects themselves. For instance, some subprojects (or portions of subprojects) may be packaged and used as dependencies. Other subprojects may consist of source repos that need to be built by the projects themselves.

We may have widely differing needs for building and packaging between different subprojects, so we will leave this up to the subprojects to decide for themselves.

**Effort and Resources:**

**Initial Contributions**:

“Base Crypto” Library (Tier 1)**:**

Our first library will be our “base crypto” library, of which the main feature is our shared modular signature library. This (work in progress) has the implementation of several different signature schemes with a common API, which allows for blockchain builders to change signature schemes almost on-the-fly (or to use and support multiple signature schemes easily). This library is being made by merging the Indy crypto library for signatures and the Sawtooth signature library. Exact implementations and APIs have not been finalized, but they are in progress.

We note that there aren’t raw crypto implementations in this library, but wrappings of code from existing libraries and also code generated by the Apache Milagro Crypto Library (AMCL).

In the future, we expect other wrappings and modular code to go in this library. For instance, some governments may require certain hash functions or random permutations (i.e. symmetric encryption schemes). While we generally recommend against using such government-mandated schemes, we want to give people that are required to do so the tools necessary, and this is something that this project will probably attempt.

The current code for this project is in Hyperledger Labs in the crypto-lib lab, available at: <https://github.com/hyperledger-labs/crypto-lib>

Z-Mix**:**

Our second initially contributed subproject will be Z-mix. Z-mix will offer a generic way to create zero knowledge proofs that prove statements about multiple cryptographic building blocks, including signatures, commitments, and verifiable encryption.

The goal of this subproject is to provide a single flexible and secure implementation to construct such zero knowledge proofs. Z-mix consists of C-callable code but there are also convenience wrappers for various programming languages.

Z-mix currently is a Hyperledger Labs project, and the code is available at: <https://github.com/hyperledger-labs/z-mix>

**Initial Maintainers:**

“Base Crypto” Library Maintainers:

* Mike Lodder (Sovrin Foundation)
* Lovesh Harchandani (Evernym)
* Dan Middleton (Intel)
* Shawn Amundson (Bitwise.io)
* Manu Drijvers (DFINITY)
* Dave Huseby (Security Maven, Linux Foundation)

Z-Mix Maintainers:

* Mike Lodder (Sovrin Foundation)
* Lovesh Harchandani (Evernym)
* Manu Drijvers (DFINITY)
* Avesta Hojjati (Digicert)

Theoretical Maintainers**:**

* Jan Camenisch (IACR fellow, cryptographer, DFINITY)
* Hart Montgomery (cryptographer, Fujitsu)
* Manu Drijvers (cryptographer, DFINITY)
* Maria Dubovitskaya (cryptographer, DFINITY)

**How to (More Practical Details):**

Meetings: We will have occasional regularly scheduled conference calls at times to be agreed upon by the set of maintainers. In addition, we will have one-time meetings as needed when issues arise. Recognizing the potentially counterproductive nature of teleconferences in excluding some participants, we will prefer to collaborate over email and github pull request comments.

**References:**

Our guiding document: <https://crypto.stanford.edu/~dabo/cryptobook/>

**Closure:**

We don’t anticipate this project disbanding. However, if we reach a point where no one is using the code in this project, or that there is minimal use of the code, then we will close the project.

**Naming:**

We haven’t decided on an official name yet for the project. We expect to consult with the marketing committee and others before firmly committing to something.

Some suggestions thus far are as follows:

1. Contour
2. Caesar
3. Ursa
4. Dan Middleton Flavored Bear Food

**Frequent Questions and Comments:**

1. What about multiple languages? An obvious issue with a shared library would be that many users might want crypto implementations in different languages. It is our opinion that building interfaces that allow for crypto implementations to be called from different languages (shims) is substantially easier than implementing (or reimplementing) secure crypto algorithms. We believe that having these shims is usually a better alternative than using multiple crypto implementations. Currently it looks like the “base language” for most of the code in the crypto-lib will be written in rust due to its C-callability (and thus portability to many other languages), but it is not certain that this will be the case in the long run.
2. Will you support multiple standards?

In the long run, our goal is to have our protocols and implementations be as modular as possible. So as long as people are willing to contribute and maintain implementations for alternate standards, then we hope to support them. This will probably take some time to implement, though, as many standardized algorithms are currently hard-coded into crypto implementations (i.e. SHA256).

1. How do we start and move toward single implementations for various protocols? Managing disputes within a cross-platform library has the potential to be quite difficult. A way to deal with this might be the following: to start, projects put their own currently used crypto algorithms in the repository. The maintainers of the individual project crypto libraries (and by corollary, the projects) own the new implementations that they copy over to the crypto library. As implementations merge (or others begin to use existing ones), who owns/maintains each implementation can be decided in an ad hoc basis by all of the parties involved. This is essentially what is happening with the Sawtooth and Indy signature libraries right now, and some of the Indy ZK code and the Z-mix initial implementations. We hope that this continues further in the crypto-lib.
2. What about different APIs? Unlike something like consensus, in general crypto APIs are very simple (and similar). Basic encryption, for instance, takes a message and a key and outputs a ciphertext (the algorithm generates randomness as well, but that doesn’t really matter here). There isn’t a lot of creativity involved in setting the API and there isn’t a lot of tangled interaction with other, blockchain-specific components like there might be in something like consensus, so, in the long run, we think that it will be easy to have a single set of APIs that make everyone happy.
3. Should the Linux Foundation require Mic Bowman to be in a dunking booth at the next Hyperledger member summit? Yes, we absolutely agree that this should happen and are all deeply concerned that it was not on the agenda for the meeting in Montreal. In fact, some of our contributors submitted a proposal for such an event (as a talk), but it was rejected by the Linux Foundation in a short sighted decision they will clearly regret in the long run.