Workgroup: Network Working Group
Internet-Draft:
draft-crypto-deployment-considerations-00
Published: 15 May 2023
Intended Status: Informational
Expires: 16 November 2023
Authors: C. A. Wood
Cloudflare

Deployment Considerations for Cryptographic Protocols

Abstract

Many real world problems require implementing and deploying cryptography as part of the solution. In general, there is no single standard or set of requirements by which applications determine what type of cryptographic solution is best for their problem. Different applications and deployments can lead to varying tradeoffs in computation, memory, network, and bandwidth properties of a solution. Moreover, practical aspects of modern software engineering, especially around long-term maintenance costs, may influence what type of cryptographic solutions are deployed in practice. This document attempts to cover different factors that influence what type of cryptography is deployed in practice with the goal of helping cryptographic researchers navigate the tradeoffs and assumptions made in new and emerging work.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/chris-wood/draft-crypto-deployment-considerations.

Status of This Memo

Table of Contents

1. Introduction
2. Conventions and Definitions
3. Functional Considerations
   3.1. Computation
   3.2. Memory
   3.3. Round Trips
   3.4. Bandwidth
4. Implementation Considerations
   4.1. Bootstrapping
   4.2. Long-Term Maintenance
5. Ecosystem and Adoption Considerations
6. General Recommendations
7. Security Considerations
8. IANA Considerations
9. References
   9.1. Normative References
   9.2. Informative References

Acknowledgments

Author's Address

1. Introduction

Software engineering is like any other form of engineering; every problem that needs a solution requires careful cost-benefit analysis to determine what type of solution is best. Naturally, there is rarely one single best answer. Engineers make tradeoffs when navigating the solution space as a way of balancing a variety of real world constraints and requirements.

Constraints and requirements vary widely in practice. For example, constraints can apply to the specific properties of a solution, such as the amount of computation or network round trips required by a particular cryptographic protocol. Constraints can also apply to the threat model and trust assumptions to a soluton, such as whether or not different parties in the system are considered trusted or not.

Finally, constraints can also apply to pragmatic engineering considerations beyond the functional properties of a cryptographic solution, such as the long-term maintenance costs of implementing a particular algorithm or solution.

Engineering is a balancing act of tradeoffs with the end goal of providing high value solutions. For problems that require (in part) cryptographic solutions, balancing the different real world constraints well is particularly important. Improper or incorrect solutions can be costly in the best case or have security vulnerabilities in the worst case. Awareness of the real world requirements or constraints that influence engineering and deployment decisions can help navigating the solution space by designing cryptographic algorithms or protocols.

To that end, the intent of this document is twofold. The first objective of this document is to discuss constraints and requirements that are factored into the implementation and deployment of real world cryptographic solutions. The second objective of this document is to survey concrete examples of deployed cryptography to illustrate how certain tradeoffs with respect to these constraints and requirements were made.

The target audience of this document is cryptographic and security researchers who are working on solutions to problems that exist in the real world.

2. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Functional Considerations

Functional constraints determine what functional properties of a solution are feasible for a particular deployment. The functional properties of a cryptographic solution generally include the amount of computation, memory, network round trips, and network bandwidth needed for a particular solution. These properties all contribute to the overall quality of a solution. For example, in applications which have real-time requirements, such as web browsing, excessive computation means that a solution takes longer to complete, which may negatively influence the user experience.

In general, functional properties affect the cost of a solution, where cost can be measured in terms of financial expense, battery

life or power consumption, or cost to the end user experience. Solutions seek to minimize their cost while maximizing their quality.

This section describes how these properties are constrained in practice to minimize cost and provides examples of how these constraints influenced other relevant quality metrics.

3.1. Computation

Minimizing computation is generally always better, but only up to a point. For example, consider the problem of authenticated key exchange with TLS. The actual key exchange step generally involves some public key cryptography operations. When used in the context of a user-facing application, such as web browser, a desired cryptographic solutions to the key exchange step should be less than the amount of time that a human can perceive any latency, as otherwise the cost of this step can negatively affect the user experience. However, minimizing the cost of this step is only useful up to the point where the cost of network round trips exceeds that of the key exchange computation. In other words, at a certain point, the cost of computation is no longer the bottleneck.

As another example, sometimes cryptographic solutions can minimize computation as a way of decreasing financial cost to deploy the solution, but this too has a limit. For example, consider the case of privacy-preserving measurement using [STAR]. STAR is a very lightweight and efficient protocol for measuring heavy hitters. However, this efficiency comes from a relaxation of the threat model. STAR is most cost efficient when it assumes honest clients and honest servers, as otherwise there is no need to do additional checks to mitigate abuse by malicious parties. STAR was motivated by the need to decrease the cost of measuring heavy hitters through less computation, yet at the cost of a weakened threat model compared to alternative solutions.

As another example, consider the case of batched zero knowledge proof generation and verification for cryptographic constructions based on VOPRFs. Solutions may wish to minimize the cost of proof generation and verification across many independent requests as a way of decreasing overall computation. However, in practice, an implementation of this solution is more complex than one in which there is no coordination across independent requests, in particular because the VOPRF signer needs to implement batching and the logic necessary to handle batch processing in a timely manner. Thus, this decrease in computation comes at the cost of implementation complexity.

3.2. Memory

Minimizing memory consumption -- or state -- is an advantageous goal. However, it's important to distinguish what types of memory or state a particular proposes when exploring this constraint. There are effectively two types of memory or state that are relevant for cryptographic solutions:

1. Long-term state, such as private keying material that exists for repeated runs of a cryptographic protocol.

2. Ephemeral state, such as one-time-use key shares and random nonces used for a key exchange protocol, that only exists for the duration of a given cryptographic protocol and is effectively captured or constrained within the state machine of a protocol.

Minimizing all three types of state or memory is generally infeasible; every solution inevitably requires some form of state. Moreover, there is generally no best answer for which type of state to minimize. Considerations that apply to each are below.

1. Long-term state. Minimizing this type of state simplifies management. For example, in the case of long-term private keys, minimizing the number of keys can simplify mechanics that need to be in place to deal with issues such as revocation and rotation. Fewer keys may need to change as a result. However, minimizing such long-term state means that each key gets more use across all possible users. In the extreme case where there is just a single key, the key becomes a shared resource with access contention. Requests to use the key can therefore lead to decreased performance, especially if access is protected by a mutex or similar. In comparison, where there is one key per possible user, this contention may not exist, but managing these keys may require more complicated key management machinery.

   Minimizing long-term state and the contention that results can also manifest in security vulnerabilities if contention is not dealt with appropriately. For example, some hash-based signature schemes require that no nonce ever be reused, as otherwise the private signing key is immediately revealed. Minimizing the number of private signing keys increases the need for contention, since if the signer does not properly track signed nonces then key compromise is possible. Some cryptographic protocols assume techniques to manage such state exist, yet for practical reasons are not very realistic. Some threshold signature protocols, for example, may assume that signature nonces are tracked globally across all signing participants, but implementing such a strongly consistent database for is

challenging in practice. The CAP (consistency, availability, partition-tolerance) theorem [TODO] strongly suggests that such assumptions are impractical for protocols that require availability, since typically consistency and partition-tolerance are necessary for security.

In general, long-term state is feasible to maintain provided there is no consequence for contention, be it a performance consequence due to mutual access or a security consequence to failure to properly manage this state.

2. Ephemeral state. Minimizing this type of state simplifies the protocol state machine. As such, minimizing this is useful insofar as it helps minimize the overall cost of the protocol. A "stateless" protocol, i.e., one in which the state machine does not encode state across rounds of interaction, is generally the best outcome, as it means that there is no need to track state across rounds of interaction. A "stateful" protocol is one in which the state machine does require state that extends beyond rounds of interaction.

   There are significant practical differences between stateful and stateless protocols. In particular, a stateless protocols maps very cleanly to stateless transport protocols such as HTTP, making them much easier to deploy in modern application environments than stateful counterparts. While stateful protocols can be implemented using HTTP as transport, this requires either storing and managing protocol state in a local database. (An alternate strategy might be to store state "on the wire," where this state is encrypted for only one party and then transferred between parties, but this requires replay protection and thefore yet again some form of local database.)

   Not all deployment environments offer some form of local database. Moreover, even for those that do, the consistency guarantee of the database may not be that which is necessary for the protocol.

3.3. Round Trips

The benefit of reducing protocol rounds depends on factors. As described in Section 3.2, minimzing the number of rounds to exactly one has the benefit of producing a stateless protocol, thereby easing deployment. Assuming all other functional characteristics (computation cost, memory, etc) stay the same, reducing the number of rounds decreases the performance profile of the protocol. However, often reducing the number of rounds negatively affects other aspects of the protocol. For example, reducing rounds may require more bandwidth per round, more complicated implementation or cryptography

in order to provide the same functionality, or it may require a weaker threat model.

As an example, some protocols are specified with a notion of preprocessing, wherein one or more parties do some amount of work a priori, either locally or in collaboration with other parties, in order to simplify the main online protocol. As an example, the [FROST] threshold signature protocol consists of two phases, one of which can be done offline by signing parties in a preprocessing phase. Splitting protocols into an offline and online phase can help reduce the cost of the online phase, but necessarily requires coordination between the offline and online phases. As described in Section 3.2, technologies for coordination may or may not be available for a particular deployment environment.

As another example, some applications of zero knowledge proofs involve an offline phase that's done to minimize the cost of proof generation in the online phase. This split is done for practical reasons: proof generation without precomputation can be expensive. However, implementations that use such precomputation are almost always more complicated in practice. Precomputation is done by some process that's running in the background and separated from processes that wish to use the output of this precomputation in the online phase. This means there must now exist some form of IPC between these processes, and, additionally, requires that the process implementing the online phase implement some form of input validation.

As a final note, there is little difference between two or more rounds in a stateful protocol from an implementation perspective. Any protocol which requires two rounds must necessarily have some mechanism for dealing with state across the rounds, and this mechanism can also be used for storing state across any subsequent rounds. Thus, practically speaking, unless there are performance reasons to do so, optimizing a protocol with more complicated cryptography to reduce the number of rounds from three or more to exactly two is often not a desirable tradeoff.

3.4. Bandwidth

Minimizing bandwidth is an important goal of a cryptographic protocol depending on the deployment environment. Some deployments have access to a transport protocol without practical bandwidth constraints. Examples of such protocols include applications built over TLS, QUIC, or HTTP. Such protocols expose streams to applications, where said streams have no practical input length restrictions. In such settings, reducing bandwidth can help improve performance by decreasing the time to transfer messages, assuming the number of rounds, computation, or memory requirements do not increase as a

result. However, in practice, protocols that have no theoretical limit may experience practical limits:

1. HTTP implementations can limit the size of headers that are used to convey information between recipients. As this is an implementation detail, the exact limit is not well established. Nevertheless, the fact that these limits exist means that protocols using HTTP in this manner must take precaution.

2. TCP segment and QUIC packet sizes have a maximum length, and sending messages that exceed the size of these lengths means that multiple packets will be sent, some of which are at the maximum size of each packet. Some networks can react poorly to such packets, e.g., by dropping them from an active connection, causing the transport protocol retranmission logic to retransmit them.

Some protocols have constraints that are imposed by the underlying protocol. For example, protocols like DNS have strict message limits codified by the wire format, meaning it is not possible to exceed these limits. Deployments of post quantum cryptographic solutions for DNSSEC, especially post quantum signatures, will face these bandwidth limits. Depending on these limits, alternative cryptographic solutions may be necessary to solve the problem. One recent example of this is [MERKLETREECERTS], wherein effectively all digital signatures in TLS handshake were replaced by compact Merkle Tree proofs.

Reducing bandwidth to avoid practical or theoretical limits is advantageous. This is especially true if the bandwidth cost exceeds the bandwidth cost of actual application data. As an example, Privacy Pass could theoretically use post quantum blind signature protocols for producing one-time-use anonymous tokens. However, in most practical applications, the size of these signatures would overshadow the size of application data that accompanies these tokens, effectively hindering deployment of the solution. In the particular case of Privacy Pass, this may mean that alternate cryptographic solutions may be required.

4. Implementation Considerations

Beyond the functional constraints that one must consider before choosing a protocol to deploy, there are also practical implementation constraints that affect deployment, including those that are paid up front to ship a solution, referred to as bootstrapping costs, and then those that are paid after the solution has been shipped.

4.1. Bootstrapping

Implementing new cryptography of any form always has a cost. For example, perhaps the new cryptography is not yet specified and exists only in academic literature. In this case, the cost of implementing it requires careful collaboration with cryptographic experts. In other cases, perhaps the cryptography is well understood and specified, but for licensing reasons it's not possible to reuse existing implementations. In this case, the cost comes from re-implementing, which comes with additional risk of introducing bugs not found in other implementations.

Even when these bootstrapping costs seem small, they may matter in practice. There are often time pressures to deliver a solution for the desired problem in a timely manner. Product deadlines push towards simple and easy-to-implement solutions, sometimes even ones with less-than-ideal security or privacy properties, over more complicated but perhaps ultimately better solutions. Cryptographic researchers can help mitigate this tradeoff in practice by working to ensure that the cost of implementing their work is minimal. That might mean simplifying protocols, producing technical specifications that faciliate engineering work, producing verified implementations that are provably correct with tools such as [hacspec], or even releasing high quality production software that could be used without license constraints.

4.2. Long-Term Maintenance

Perhaps the most important implementation factor to consider when choosing a cryptographic solution is the long-term maintenance cost. There are many factors that go into this cost.

1. External dependencies. Implementing a cryptographic solution that is exposed to users through an API or some other way means that implementations must continue to maintain and update this functionality over time. If there are bugs or security vulnerabilities, they must be fixed. External dependencies of this nature tend to inevitably ossify in practice, meaning that it is difficult to transition users off the solution and towards something better. Cryptographic solutions with no external users are much easier to manage long term as they do not require coordination or a more thoughtful deprecation strategy.

   It's important to note that external dependencies may not always come in the form of a user calling some new API. Such dependencies can be through high-level features that are built on the cryptographic solution. For example, imagine a contact discovery application built on some version of private information retrieval (PIR). The dependents of the PIR

technology are not necessarily applications that invoke the system, but the end users themselves, who come to expect certain privacy properties for contact discovery. New cryptographic solutions may require new conceptual mental models for users that they are unfamiliar with, making it difficult for end users to meaningfully engage with the system. As a relevant example, it seems unclear how one would succinctly explain concepts like multiparty computation to a layperson. The cost of educating and informing such users about relevant details of the system should be factored in where appropriate.

Another relevant example is the deployment of cryptographic solutions with no obvious post quantum upgrade path. Consider the design of non-trivial anonymous credential systems built on pairing-friendly curves, e.g., such as those built with [BBS] and [BLS] under the hood. Pairings do not currently have a post-quantum variant. As such, any application that deploys these anonymous credentials potentially introduces a new feature for users that cannot be replicated in the advent of a post quantum capable attacker. Deploying technology with such limitations in place ultimately does a disservice to the end user.

2. Internal dependencies. Implementing new cryptographic solutions that pull in new cryptographic dependencies is also problematic. Depending on the platform and deployment environment, new dependencies can be problematic for a number of reasons. They may introduce new code into production, thereby expanding the attack surface. Auditing or verifying these dependencies can help, but is imperfect. New dependencies can lead to code and binary bloat, especially with cryptographic implementations that do not themselves minimize dependencies. Finally, new internal dependencies mean that these dependencies must be actively maintained going forward. If there are bugs or security vulnerabilities reported, they must be patched.

3. Knowledge cost. Sometimes the engineers that bringup a solution are not those that maintain a solution long-term. This is often true when, for example, production services transfer from an engineering team actively building a service to a service reliability team that maintains it. Successful maintenance of a cryptographic solution generally requires some working knowledge of this system in order to ease SRE costs, and there is a cost in transferring this knowledge across teams. This cost is paid in the development of training, education, and documentation resources, for example. This cost can be paid through the natural course of product development, or other change in product ownership, e.g., if an knowledgeable engineer leaves the project or the project is transferred to another team.

For these reasons, cryptographic solutions that minimize new dependencies, reuse existing cryptographic components, and are accommodating to future threat models (such as post quantum attackers) are highly desirable and, in practice, will often eclipse solutions that offer only slightly better functional properties. The long-term maintenance costs are bearable provided the proposed cryptographic solution has noticably better functional, security, or privacy properties.

5. Ecosystem and Adoption Considerations

Constraints and considerations for cryptographic solutions also extend to the ecosystem in which they are deployed. Cryptographic solutions that solve many diverse problems are generally better than those which are single purpose. As a practical example, iCloud Private Relay and Private Access Tokens both use blind RSA [BLINDRSA] as a form of anonymous credential in part because the same protocol could fit multiple use cases. In fact, blind RSA has been proposed as a solution to more use cases in practice, including those around anti-abuse for privacy-preserving ad click attribution. It would have been possible to use VOPRFs for iCloud Private Relay and blind RSA for Privacy Pass for performance reasons, but implementing fewer cryptographic primitives offered more value than slight improvements in functional properties.

Beyond solution reuse, there is also value in implementation reuse. Using blind RSA as another example, this particular protocol only requires the client and signer to support the protocol; it does not require the verifier of the protocol to implement any new cryptography, since most popular cryptographic libraries support RSA signature verification. This meant that use of blind RSA for these two use cases described above eased adoption for consumers of blind RSA tokens by not forcing any new cryptographic solutions onto verifiers.

Cryptographic solutions sometimes also make inaccurate assumptions about deployment environments, or fail to factor in properties of existing deployments. For example, some cryptographic protocols are designed with specific algorithms baked into the specification, sometimes because these algorithms offer better security properties, but these algorithms do not have widespread deployment in existing infrastructure. As an example, most hardware software modules (HSMs) are quite limited in the types of algorithms they support. For instance, some only support a variety of NIST curves. As such, in deployments where this limitation exists, cryptographic solutions that require an elliptic curve as the basis for a prime-order group will lean towards solutions built on NIST curves, as doing so accommodates existing deployed infrastructure.

6. General Recommendations

This document discusses a number of considerations and constraints that influence how engineering decisions are made. Again, there is certainly no single set of standard best practices or recommendations that would guide towards a consistent outcome. However, there are very general practices that do empirically yield positive outcomes for the deployment of cryptographic solutions. This section discusses some of these recommendations.

1. Always favor simplicity. Simplicity will almost always yield the best outcomes. Simple protocols and algorithms are easier to understand, implement, and ultimately deploy. Sometimes we pay for simplicity with lack of generality or forward-looking support, or by lesser functional properties, but these costs are almost always worth the gains that come from a conceptually simple protocol.

2. Maximize reuse where possible, especially when avoiding reuse only leads to marginal gains. Reuse typically means less work for implementers to adopt, communicate, and maintain a solution long term.

3. Collaborate. When in doubt, maintain open lines of communication and collaboration with industry to help navigate the many dimensions of tradeoffs discussed in this document. This isn't always feasible, especially for industry sectors that work in isolation or secret, but where feasible, leverage the opportunity.

7. Security Considerations

This document discusses considerations relevant to the implementation and deployment of cryptography in practice. The document is effectively a collection of security considerations.

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/

RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

9.2. Informative References

[BBS] Looker, T., Kalos, V., Whitehead, A., and M. Lodder, "The BBS Signature Scheme", Work in Progress, Internet-Draft, draft-irtf-cfrg-bbs-signatures-02, 11 March 2023, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bbs-signatures-02>.

[BLINDRSA] Denis, F., Jacobs, F., and C. A. Wood, "RSA Blind Signatures", Work in Progress, Internet-Draft, draft-irtf-cfrg-rsa-blind-signatures-12, 3 April 2023, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-rsa-blind-signatures-12>.

[BLS] Boneh, D., Gorbunov, S., Wahby, R. S., Wee, H., Wood, C. A., and Z. Zhang, "BLS Signatures", Work in Progress, Internet-Draft, draft-irtf-cfrg-bls-signature-05, 16 June 2022, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bls-signature-05>.

[FROST] Connolly, D., Komlo, C., Goldberg, I., and C. A. Wood, "Two-Round Threshold Schnorr Signatures with FROST", Work in Progress, Internet-Draft, draft-irtf-cfrg-frost-13, 8 May 2023, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-frost-13>.

[hacspec] "hacspec: A specification language for crypto primitives in Rust", n.d., <https://github.com/hacspec/hacspec>.

[MERKLETREECERTS] Benjamin, D., O'Brien, D., and B. Westerbaan, "Merkle Tree Certificates for TLS", Work in Progress, Internet-Draft, draft-davidben-tls-merkle-tree-certs-00, 10 March 2023, <https://datatracker.ietf.org/doc/html/draft-davidben-tls-merkle-tree-certs-00>.

[STAR] Davidson, A., Sahib, S. K., Snyder, P., and C. A. Wood, "STAR: Distributed Secret Sharing for Private Threshold Aggregation Reporting", Work in Progress, Internet-Draft, draft-dss-star-02, 24 October 2022, <https://datatracker.ietf.org/doc/html/draft-dss-star-02>.

Acknowledgments

This document was motivated by discussions that took place during the Real World Crypto 2023 conference.

Author's Address

Christopher A. Wood
Cloudflare

Email: caw@heapingbits.net