Withdrawn Draft

Warning Notice

The attached draft document has been withdrawn and is provided solely for historical purposes. It has been followed by the document identified below.

Withdrawal Date July 17, 2025

Original Release Date March 5, 2025

The attached draft document is followed by:

Status Second Public Draft (2pd)

Series/Number NIST CSWP 39 2pd

Title Considerations for Achieving Cryptographic Agility: Strategies and Practices

Publication Date July 17, 2025

DOI https://doi.org/10.6028/NIST.CSWP.39.2pd

CSRC URL https://csrc.nist.gov/pubs/cswp/39/considerations-for-achieving-cryptographic-agility/2pd

Additional Information

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

Check for updates

NIST Cybersecurity White Paper
NIST CSWP 39 ipd

Considerations for Achieving Crypto Agility

Strategies and Practices

Initial Public Draft

Elaine Barker*
Lily Chen
David Cooper
Dustin Moody
Andrew Regenscheid
Murugiah Souppaya*
Computer Security Division
Information Technology Laboratory

**Former NIST employee; all work for this publication was done while at NIST.*

Bill Newhouse
Applied Cybersecurity Division
Information Technology Laboratory

Russ Housley
Vigil Security

Sean Turner
sn3rd

This publication is available free of charge from:
https://doi.org/10.6028/NIST.CSWP.39.ipd

March 5, 2025

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

Certain equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification does not imply recommendation or endorsement of any product or service by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

NIST Technical Series Policies
Copyright, Use, and Licensing Statements
NIST Technical Series Publication Identifier Syntax

Publication History
Approved by the NIST Editorial Review Board on YYYY-MM-DD [Will be added to final publication.]

How to Cite this NIST Technical Series Publication:
Barker E, Chen L, Moody D, Regenscheid A, Souppaya M, Newhouse B, Housley R, Turner S (2025) Considerations for Achieving Crypto Agility: Strategies and Practices. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Cybersecurity White Paper (CSWP) NIST CSWP 39 ipd. https://doi.org/10.6028/NIST.CSWP.39.ipd

Author ORCID iDs
Elaine Barker: 0000-0003-0454-0461
Lily Chen: 0000-0003-2726-4279
David Cooper: 0000-0001-2410-5830
Dustin Moody: 0000-0002-4868-6684
Andrew Regenscheid: 0000-0002-3930-527x
Murugiah Souppaya: 0000-0002-8055-8527
Bill Newhouse: 0000-0002-4873-7648

Public Comment Period
March 5, 2025 - April 30, 2025

Submit Comments
crypto-agility@nist.gov

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930

Additional Information
Additional information about this publication is available at https://csrc.nist.gov/publications/cswp, including related content, potential updates, and document history.

All comments are subject to release under the Freedom of Information Act (FOIA).

Abstract

Cryptographic (crypto) agility refers to the capabilities needed to replace and adapt cryptographic algorithms in protocols, applications, software, hardware, and infrastructures without interrupting the flow of a running system in order to achieve resiliency. This white paper provides an in-depth survey of current approaches to achieving crypto agility. It discusses challenges and tradeoffs and identifies approaches for providing operational mechanisms to achieve crypto agility while maintaining interoperability. It also highlights critical working areas that require additional discussion.

Keywords

cryptographic agility; cryptographic transition; cryptography; interoperability; security protocol.

Audience

This white paper's intended audience includes protocol designers, IT administrators, software and standards developers, hardware designers, and policymakers. Achieving crypto agility includes proactively addressing upcoming transitions and ensuring that the issues highlighted will capture the attention of cryptographic researchers.

Note to Reviewers

The goal of this draft white paper is to establish a common understanding of challenges and identify existing approaches related to crypto agility, based on the discussions that NIST has conducted with various organizations and individuals. This paper serves as read-ahead material for an upcoming NIST-hosted virtual workshop where crypto agility considerations will be discussed with the cryptographic community to further identify future areas of work and inform the development of the final paper.

Acknowledgments

NIST appreciates the input and contributions from the collaborators of the National Cybersecurity Center of Excellence (NCCoE) Post-Quantum Cryptography (PQC) Migration project, who commented on an initial version of this paper and provided comments that were considered in the development of this draft white paper. A special note of thanks goes to Karen Scarfone, Jim Foti, and Isabel Van Wyk for reviewing and editing this document.

Table of Contents

1. Introduction 1
2. Historic Transitions and Challenges 3
2.1. Long Period for a Transition 3
2.2. Backward Compatibility and Interoperability Challenges 3
2.3. Constant Needs of Transition 4
2.4. Resource and Performance Challenges 4
3. Crypto Agility for Security Protocols 5
3.1. Algorithm Identification 5
3.1.1. Mandatory-to-Implement Algorithms 6
3.1.2. Dependent Specifications 7
3.2. Algorithm Transitions 7
3.2.1. Preserving Interoperability 8
3.2.2. Providing Notices of Expected Changes 9
3.2.3. Integrity for Algorithm Negotiation 9
3.2.4. Hybrid Cryptographic Algorithms 10
3.3. Cryptographic Key Establishment 11
3.4. Balancing Security Strength and Protocol Complexity 12
3.4.1. Balancing the Security Strength of Algorithms in a Cipher Suite 12
3.4.2. Balancing Protocol Complexity 12
4. Crypto Agility for Applications 13
4.1. Using an API in a Crypto Library Application 13
4.2. Using APIs in the Operating System Kernel 14
4.3. Hardware 15
5. Discussions 17
5.1. Resource Considerations 17
5.2. Agility Awareness Design 17
5.3. Complexity and Security 18
5.4. Crypto Agility in the Cloud 18
5.5. Maturity Assessment for Crypto Agility 19
5.6. Crypto Agility Strategic Plan for Managing Organizations' Crypto Risks 20
5.6.1. Crypto Standards, Regulations, and Mandates 21
5.6.2. Crypto Security Policy Enforcement 22
6. Conclusion 23
References 24

Appendix A. List of Symbols, Abbreviations, and Acronyms 27

List of Figures

Fig. 1. Possible Second Transition from Hybrid Mode 10
Fig. 2. Applications Using Crypto APIs 14
Fig. 3. Crypto Agility Strategic Plan for Managing Organizations' Crypto Risks 20

1. Introduction

Advances in computing capabilities, cryptographic research, and cryptanalytic techniques periodically create the need to replace algorithms that no longer provide adequate security for their use cases with algorithms that are considered secure. For example, the threats posed by future cryptographically relevant quantum computers (CRQCs) to public-key cryptography demand an urgent migration to quantum-resistant cryptography. Such a transition is costly and takes time, raises interoperability issues, and disrupts operations.

Cryptographic (crypto) agility describes the capabilities needed to replace and adapt cryptographic algorithms for protocols, applications, software, hardware, and infrastructures without interrupting the flow of a running system in order to achieve resiliency. Properly designed operational mechanisms that incorporate crypto agility considerations are needed to facilitate transition to newer algorithms in a fast and smooth way without introducing security breaches or operational disruptions. Many definitions and descriptions of crypto agility have been proposed. For example, a 2016 NIST presentation [1] described crypto agility as:

- The ability for machines to select their security algorithms in real time and based on their combined security functions;
- The ability to add new cryptographic features or algorithms to existing hardware or software, resulting in new, stronger security features; and
- The ability to gracefully retire cryptographic systems that have become either vulnerable or obsolete.

In the proposed definition, crypto agility is described as an algorithm-agnostic ability to support multiple cryptographic algorithms in systems, protocols, software, and hardware. Crypto agility facilitates migrations between cryptographic algorithms without significant changes to the application that is using the algorithms. Crypto agility must be considered for each specific implementation environment. In this white paper, we provide general considerations for crypto agility within the context of a computing platform, a protocol, and an enterprise IT system.

Cryptographic algorithms are implemented in software and hardware to facilitate their use in applications. For example, replacing a cryptographic algorithm in applications will require changes to application programming interfaces (APIs) and software libraries. It may also necessitate the replacement of hardware to incorporate new hardware accelerators. *In a system, crypto agility is the ability to adopt new cryptographic algorithms and stop the use of weak algorithms in applications without disruptions to the running system*.

In a communication protocol, parties must agree on a common *cipher suite*, a set of cryptographic algorithms used for key establishment, signature, hash function, encryption, and/or data authentication. Any update of algorithms must be reflected in the protocol specifications. *In a protocol, crypto agility is the ability to maintain interoperability when introducing new cryptographic algorithms and preventing the use of weak algorithms*.

Achieving crypto agility is not only a task for product designers or implementors but also for practitioners, security policy makers, and IT administrators. Organizations that practice crypto agility should be able to turn off the use of weak cryptographic algorithms quickly when a vulnerability is discovered and adopt new cryptographic algorithms without making significant changes to infrastructures and without suffering from unnecessary disruptions.

Achieving crypto agility requires a systems approach to providing mechanisms that enable transition to a new algorithm in a seamless way while maintaining security and acceptable operation. This white paper surveys crypto agility approaches in different implementation environments and proposes strategies for achieving the agility needs of varied applications. This paper also discusses crypto agility in different contexts and highlights the coordination needed among stakeholders. The purpose of the paper is to identify critical working areas that will be discussed in an upcoming workshop to determine future work needed for achieving crypto agility.

The paper is structured as follows. Section 2 discusses historical challenges faced in past transitions. Section 3 provides an overview of an approach to achieving crypto agility for security protocols to start the discussion. Section 4 addresses strategies for supporting crypto agility in a system — from an API to software libraries or hardware. Some of them have been implemented in today's systems, and others are for future consideration. Section 5 discusses tradeoffs with crypto agility and identifies some areas for future work. Section 6 provides concluding thoughts.

2. Historic Transitions and Challenges

The security of cryptographic algorithms is constantly challenged by increases in computing power and the sophistication of cryptanalytic techniques. As a result, cryptographic transitions to replacement algorithms have become an important part of security practice.

In the past 50 years, applications involving cryptography have undergone multiple transitions. This section summarizes the transition challenges experienced and the lessons learned. In the historic review, necessary background on cryptographic algorithms and transition triggers is provided to help readers with subsequent content in this paper.

2.1. Long Period for a Transition

In 1977, Data Encryption Standard (DES) became the first published encryption standard. The DES algorithm [2] had a 64-bit block size and a 56-bit key. Motivated by the threat of a practical brute-force attack against DES's 56-bit key, Triple DES [3] (due to its capacity to use two or three 56-bit keys) was introduced as a temporary solution before a stronger algorithm could be standardized and made available for use. Though this stronger algorithm, called Advanced Encryption Standard (AES) [3] (with options for 128-, 192-, or 256-bit keys), was standardized in 2001, Triple DES only became disallowed in 2024. This 23-year transition from Triple DES to AES supports the existence of significant transition challenges.

Historically, decisions on the choice of cryptographic algorithms used for applications were made without considering any future transitions. Sometimes, the algorithms are hard coded — that is, the cryptographic algorithm is directly written into the source code of the application. It is fixed and cannot be easily changed without modifying the code itself, and it is harder to maintain and update with new algorithms.

2.2. Backward Compatibility and Interoperability Challenges

The need for backward compatibility can also be a barrier to transition. For example, hash functions are used as a message digest in digital signatures, for the generation of message authentication codes (MACs), for key-derivation functions, and for random-number generation. Cryptographic hash functions have also been used as a basic component in hash-based signatures. Cryptographic hash function requirements include collision resistance, pre-image resistance, and second pre-image resistance. SHA-1, a hash function with a 160-bit output length [4], was expected to provide 80 bits of collision resistance and 160 bits of pre-image resistance. Many use cases relied on these security properties. However, in 2005, SHA-1 was found to provide fewer than 80 bits of collision resistance [5]. In 2006, NIST responded by urging federal agencies to "stop relying on digital signatures that are generated using SHA-1 by the end of 2010."

Because SHA-1 has been used in signatures for entity authentication in many existing secure protocols, interoperability and backwards compatibility must be considered in the transition. In particular, using SHA-1 in this way had to be allowed in certain circumstances for some protocols such as Transport Layer Security (TLS) (Section 4.4.2.2 of [6]). Since 2005, additional

cryptanalyses have shown the weakness of SHA-1 with respect to not only collision resistance but also pre-image and second pre-image resistance [7]. NIST has recommended a complete transition away from SHA-1 for any usage by the end of 2030 [8]. This example shows that when some applications do not have crypto agility and cannot make timely transitions, for backward compatibility a weak algorithm has to be allowed longer than it should be.

2.3. Constant Needs of Transition

For a public-key cryptographic algorithm, security strength is determined by parameter selection. For example, one of the parameters for the RSA algorithm is the modulus size. When the use of RSA was first approved for digital signatures in 2000 as specified in Federal Information Processing Standard (FIPS) 186-2 [9], a minimum modulus size of 1024 bits was required to provide at least 80 bits of security strength. In 2013, the minimum modulus was increased to 2048 bits to provide a security strength of at least 112 bits, due to the progress in integer factorization and the increase in computing power. The transition to a larger key size (modulus) may need to happen during a device's lifetime. If a device is not designed to transition to a larger key size (modulus) during its lifetime, it will need to be replaced. Given the long lifespan of many devices, it is generally more cost-effective to design for such transitions from the start.

Since 2005, NIST Special Publication (SP) 800-57 Part 1 [10] has projected the need to transition to 128-bit security strength by 2031. In 2024, NIST Internal Report (IR) 8547 [11] stated that the 112-bit security strength for the current public-key algorithms would be deprecated in 2031 in order to facilitate a direct transition from the 112-bit security strength provided by current public-key schemes to post-quantum cryptography, without an intermediate transition to the 128-bit security strength for the current cryptographic schemes.

2.4. Resource and Performance Challenges

Transitions in general and transitions to post-quantum algorithms in particular present many challenges. Some algorithm parameter sets will have larger sizes for public keys, signatures, and ciphertext than those used previously. For example, an RSA modulus of 3072 bits provides 128 bits of security strength for its 3072-bit signature. The transition to the post-quantum Module-Lattice-Based Digital Signature Algorithm (ML-DSA) specified in FIPS 204 will result in a signature of 2420 bytes (i.e., 19,360 bits) to provide a roughly equivalent classical security strength of 128 bits [12]. This shows that transition to new algorithms can challenge the capacity of a communication network and increase the time to transmit the message with signatures or ciphertexts.

In summary, the many issues that arise during a cryptographic transition cause the transition period to be incredibly long, often longer than planned. This document intends to illustrate how crypto agility can provide a lens through which cryptographic transitions are planned and executed as part of a design and implementation plan.

3. Crypto Agility for Security Protocols

Many security protocols use cryptographic algorithms to provide confidentiality, integrity, authentication, and/or non-repudiation. Communicating peers must agree on a common set of cryptographic algorithms, referred to as a *cipher suite,* for security protocols to work properly. This aspect of a security protocol is called *cipher suite negotiation*. The cipher suite may include algorithms for integrity protection, authentication, key derivation, key establishment, encryption, and digital signatures to provide the needed security services. Crypto agility is achieved when a security protocol can easily transition from one cipher suite to another, more desirable one. Each security protocol normally specifies a mandatory-to-implement algorithm to ensure that basic interoperability is supported.

To achieve crypto agility, security protocol implementations should be modular to easily accommodate the insertion of new algorithms or cipher suites. Implementations should also provide a way to determine when deployed implementations have shifted from the old algorithms to the more desirable ones. Crypto agility means that a security protocol must support one or more algorithm or cipher suite identifiers, with the expectation that the set of mandatory-to-implement algorithms will change over time.

This section discusses challenges and existing practices in achieving crypto agility for security protocols.

3.1. Algorithm Identification

Security protocols include a mechanism to identify the algorithm or cipher suite in use. Some security protocols explicitly carry algorithm identifiers or a cipher suite identifier, while others rely on configuration settings to identify the algorithms or cipher suite. For example, an entry in a database of symmetric keys that includes a key value as well as an algorithm identifier might be sufficient. If a security protocol does not carry an explicit algorithm identifier, a new protocol version number or some other major change is needed to transition to a new algorithm or cipher suite.

The version number of a protocol or an algorithm identifier is needed for an implementation to tell communicating peers to use a different algorithm or cipher suite. Thus, crypto agility is easier to achieve when security protocols include algorithm or cipher suite identifiers.

In some security protocols, a combination of the protocol version number and explicit algorithm or cipher suite identifiers is defined. For example, in TLS version 1.2 (TLSv1.2) [13] and TLS version 1.3 (TLSv1.3) [6], the version number specifies the default key derivation function, and the cipher suite identifier specifies the other algorithms.

Some security protocols carry one identifier for each algorithm that is used, while other security protocols carry one identifier for a cipher suite that specifies the use of multiple algorithms. For example, in the IPsec protocol suite, Internet Key Exchange Protocol version 2 (IKEv2) [14] most commonly negotiates algorithms with a separate identifier for each algorithm. In contrast, TLSv1.3 [6] negotiates algorithms with cipher suite identifiers. Both identification approaches

are used successfully in security protocols, and both require the assignment of new identifiers to add support for new algorithms.

Designers are encouraged to pick one of these approaches and use it consistently throughout the protocol or family of protocols. Cipher suite identifiers make it easier for the protocol designer to avoid incomplete specifications. However, cipher suite identifiers inherently face a combinatoric explosion as all useful combinations of algorithms are specified. On the other hand, algorithm identifiers impose a burden on implementations to determine, during session establishment, which algorithm combinations are acceptable. This determination is often a negotiation that is built into session establishment, which is sometimes called *security association establishment*.

Regardless of the mechanism used, security protocols historically negotiate the symmetric cipher and cipher mode together to ensure that they are compatible. As a result, one algorithm identifier names both the symmetric cipher and the cipher mode.

In some protocols, the length of the key to be used is not specified by the algorithm or cipher suite identifier. For example, TLSv1.2 cipher suites include Diffie-Hellman key exchange without specifying a particular public-key length. If the algorithm identifier or suite identifier specifies a particular public-key length, migration to longer lengths would require the specification, implementation, and deployment of a new algorithm or cipher suite identifier. On the other hand, a flexible public-key length in a cipher suite would make it easier to migrate away from short key lengths when the computational resources available to an attacker dictate the need to do so. However, the flexibility of asymmetric key lengths has led to interoperability problems. To avoid these interoperability problems in the future, any aspect of the algorithm not specified by the algorithm identifiers needs to be negotiated, including the key size and other parameters.

3.1.1. Mandatory-to-Implement Algorithms

For secure interoperability, communicating peers must agree on a common set of secure cryptographic algorithms. While many algorithms are often specified for a security protocol, an implementation may not support all of the possible algorithms. To ensure that interoperation is possible for all implementations, a standards developing organization (SDO) will often select at least one set of strong algorithms to be mandatory to implement.

However, SDOs need to change the set of mandatory-to-implement algorithms over time to keep up with advances in computing and cryptanalysis. For example, NIST has withdrawn approval for the DES encryption algorithm, the Triple DES encryption algorithm, and the SHA-1 hash function. Each of these was a mandatory-to-implement algorithm in various security protocols at one time. It is highly desirable for SDOs to be able to revise mandatory-to-implement algorithms without modifying the base security protocol specification. To achieve this goal, some SDOs publish a base security protocol specification and a companion document describing the supported algorithms, allowing the update of one document without necessarily modifying the other.

SDOs should specify the new algorithms before the current ones have weakened to the breaking point. For example, support for the AES algorithm was introduced in S/MIME v3.1 [15], and the AES algorithm became mandatory-to-implement in S/MIME v3.2 [16]. This approach allows a timely migration to the new algorithms while the old algorithms are still able to meet their security expectations. However, a failure of implementers and administrators to take prompt action will increase the period of time that an old algorithm is used, perhaps dangerously so.

3.1.2. Dependent Specifications

Mandatory-to-implement algorithms are not specified for protocols embedded in other protocols; in these cases, the higher-level protocol specification identifies the mandatory-to-implement algorithms used in the embedded protocols. For example, S/MIME version 3.2 [16] (a higher-level protocol) makes use of (embeds) the cryptographic message syntax (CMS) [17]; thus, S/MIME (not CMS) specifies the mandatory-to-implement algorithms. This approach allows various security protocols to use CMS and make independent choices regarding which algorithms are mandatory to implement.

To add a new algorithm, the conventions for using that new algorithm are specified for the embedded security protocol (CMS in the example above), and then at some future time, the higher-level protocol (S/MIME in the example above) might make that algorithm mandatory to implement.

3.2. Algorithm Transitions

Transition from a weakening algorithm can be complicated. It is relatively straightforward to specify how to use a new, better algorithm. However, the security protocol specification, implementation development, and deployment often take years, especially if new or additional infrastructure is required prior to deployment. The physical location of devices can add challenges to upgrades, especially for remote sensors and space systems; overcoming these challenges takes time and increases cost. Then, when the new algorithm is widely deployed, the old algorithm should no longer be in use. However, knowledge about the actual use of the new algorithm will always be imperfect, so one cannot be completely sure it is safe to remove the old algorithm from an implementation.

Algorithm transition is naturally facilitated as part of an algorithm selection or negotiation mechanism. During the negotiation phase, security protocols traditionally select the most secure algorithm or cipher suite supported by all communicating peers and acceptable by their policies. In addition, a mechanism to determine whether a new algorithm has been deployed is often needed. For example, the SMIMECapabilities attribute [16] allows S/MIME mail user agents to share the list of algorithms they are willing to use in order of preference. A secure email sender can tell that it is possible to use a new algorithm when all recipients include it in their SMIMECapabilities attribute. As another example, the Extension Mechanisms for DNS (EDNS(0)) [18] can be used in Domain Name System Security Extensions (DNSSEC) to signal the acceptance and use of new digital signature algorithms. In the Resource Public Key

Infrastructure (RPKI), all implementations must support the same digital signature algorithm. To ensure global acceptance of a digital signature, an approach to transition has been specified where a new signature algorithm is introduced long before the original one is phased out [19].

In the worst case, a deeply flawed algorithm may still be available and used in an implementation, which could permit an attacker to download a simple script to compromise the data that the algorithm is intended to protect. Sadly, flawed security can also occur when a secure algorithm is used incorrectly or used with poor key management. In such situations, it is not possible to provide notice to implementers as discussed in Sec. 3.2.2, and the protection offered by the algorithm is severely compromised, perhaps to the point that administrators want to stop using the weak cipher suite that includes the algorithm altogether, rejecting offers to use the weak cipher suite well before the new cipher suite is widely deployed.

In any case, there comes a point in time when administrators configure their implementations to refuse the old, weak crypto suite. This can happen by picking a date for a global switch to the new algorithm, or each installation can select a date on their own. In either case, interoperability will be sacrificed with any implementation that does not support the new crypto suite.

3.2.1. Preserving Interoperability

Removing support for deprecated and obsolete cryptographic algorithms is very challenging. Once an algorithm is determined to be weak, it is very difficult to eliminate all uses of that algorithm because many applications and environments rely on it. Since algorithm transitions can introduce interoperability problems, protocol designers and implementers may be inclined to delay the removal of support for algorithms. As a result, flawed algorithms can be supported for far too long. The security impact of using legacy software that includes the flawed algorithm and having extended support periods can be reduced by making algorithm transitions easy. Social pressure is often needed to cause the transition to happen. For example, the RC4 stream cipher was supported in web browsers until Andrei Popov championed an effort to stop its use [20].

Implementers are often reluctant to remove deprecated algorithms from server software, and server administrators are often reluctant to disable them over concerns that some party will no longer have the ability to connect to their server. Implementers and administrators want to improve security by using the strongest supported algorithms, but their actions are tempered by the desire to preserve interoperability. Some web browsers provide a visual warning when a deprecated algorithm is selected for use. These visual warnings provide an incentive for website operators to transition away from deprecated algorithms.

Transition in the internet infrastructure is particularly difficult. The digital signature on a certification authority (CA) [21] certificate is often expected to last decades, which hinders transition away from a weak signature algorithm. Once a long-lived certificate is issued with a particular signature algorithm, that algorithm is used by many relying parties to verify certificates signed by the CA, and none of the relying parties can stop supporting it without

invalidating all of the certificates signed by that CA. Many certificates can be impacted by the decision to drop support for a weak signature algorithm or an associated hash function; all subjects need to get new certificates.

Influential organizations such as NIST and the Internet Engineering Task Force (IETF) can assist with overcoming the conflicting desire to preserve interoperability by coordinating the deprecation of an algorithm or cipher suite, simplifying the transition for their own users as well as others.

3.2.2. Providing Notices of Expected Changes

Fortunately, cryptographic algorithm failures without warning are rare. Algorithm transitions are typically driven by advancements in computing capabilities, cryptographic research, and cryptanalytic techniques rather than unexpected failures. For example, the transition from DES to Triple DES to AES took place over decades, resulting in a shift in symmetric block cipher security strength from 56 bits to 112 bits to at least 128 bits. Where possible, SDOs should provide notice to security protocol implementers about expected algorithm transitions.

Monitoring cryptographic research results provides a way to assess impact and foresee needed changes. The cryptographic research community might discover a new attack with practical impact to existing security protocols. In the worst case, a breakthrough cryptanalytic technique can indicate the need for an immediate algorithm transition. Crypto agility is needed to smoothly implement such a transition.

Looking forward to the transition to post-quantum cryptography (PQC), security protocol designers need to plan, as part of their crypto agility efforts, for public keys, signatures, and key-encapsulation ciphertext to be much larger than those currently used. Of course, public-key sizes and signature sizes directly impact the size of certificates containing those keys and signatures. To be safe, security protocol designers should plan for a growth of at least ten-fold based on the key sizes for classical algorithms and PQC algorithms.

3.2.3. Integrity for Algorithm Negotiation

Cryptographic algorithm selection or negotiation should have its integrity protected. If the integrity of algorithm selection during negotiation is not protected, the protocol will be subject to a downgrade attack, where an attacker influences the choice of cipher suite and one with weaker algorithms is chosen. Transition mechanisms need to consider the algorithm that is used to provide integrity protection for algorithm negotiation. If a protocol specifies a single integrity algorithm to protect the negotiation without a way to negotiate an alternative integrity algorithm, eventually that single algorithm will be found to be weak.

Extra care is needed when a mandatory-to-implement algorithm is used to provide integrity protection for the negotiation of other cryptographic algorithms. In this case, the integrity protection should be at least as strong as that provided by the next set of algorithms, which can result in the need for several mandatory-to-implement algorithms to cover the various security

strength requirements. Otherwise, a flaw in the mandatory-to-implement integrity algorithm may allow an attacker to influence the choices of the other algorithms.

Security protocols can negotiate a key-establishment mechanism, derive an initial cryptographic key, and then authenticate the negotiation. However, if the authentication fails, the only recourse is to start the negotiation over from the beginning. This is necessary for security but can lead to an awkward experience for the human user when authentication is unsuccessful.

3.2.4. Hybrid Cryptographic Algorithms

The transition from traditional to quantum-resistant public-key cryptographic algorithms is underway, and some SDOs are considering a combination of the two types of public-key algorithms to create a hybrid algorithm.[1] The idea is to continue using the well-tested traditional algorithms while study of the new PQC continues and the implementations are maturing. In most cases, choosing a hybrid algorithm leads to a second transition when the traditional algorithm is deprecated, as shown in Fig. 1.



Fig. 1. Possible second transition from hybrid mode

Some people believe that the overhead associated with the traditional algorithm is small enough that they will avoid the second transition. That is, these people will continue to use the hybrid algorithm even when the traditional algorithm is no longer secure.

A hybrid signature algorithm combines a traditional signature algorithm, like Elliptic Curve Digital Signature Algorithm (ECDSA), and a PQC signature algorithm, like ML-DSA [12]. A hybrid signature algorithm requires that two public keys be certified: a public key for the traditional algorithm and a PQC public key. One option is to include the two public keys in a single certificate, where the public keys would always be used together. However, the cost of deploying a PKI root of trust is significant, so the expense associated with a transition to the use of a hybrid root of trust followed by a second transition to using only a PQC algorithm for a root of trust must be considered.

Another option is the deployment of a traditional root of trust and a PQC root of trust using separate certificates. In some cases, two certificates will be less expensive, but there are operational costs associated with validating two certification paths for security-association establishment. A significant advantage of using separate roots of trust is that once the traditional PKI is no longer needed, one can simply stop issuing certificates under the traditional root of trust, while the PQC trust anchor continues to be used. Simply let it wither. Of course, the PQC root of trust continues to be used.

A hybrid key-establishment algorithm combines the use of a traditional key-establishment algorithm, like Diffie-Hellman key exchange specified in SP 800-56A [22], and the use of a PQC

[1] Some of the hybrid algorithm specifications refer to “composite algorithms.” At the level of the discussion in this section, the distinctions between “hybrid” and “composite” algorithms are unimportant. Thus, this section uses “hybrid” throughout.

key-encapsulation mechanism (KEM), like the Module-Lattice-Based Key Encapsulation Mechanism (ML-KEM) [23], to establish a pairwise shared secret under the assumption that at least one of the algorithms will remain strong over time. Security analysis for a hybrid key-establishment algorithm can be more complicated than the analysis of either of the algorithms that are used in the hybrid algorithm. In addition, the use of hybrid key-establishment algorithms increases bandwidth usage because more data needs to be exchanged, which can be a problem for some implementation environments.

In summary, hybrid signatures or key-establishment schemes can be a good strategy for preserving security in the face of uncertainty while transitioning from traditional public-key cryptography to post-quantum cryptography, but the use of hybrid schemes increases protocol complexity and the amount of resources consumed. Hybrid signatures or key-establishment schemes exercise the capability to accommodate many cipher suites and stress the crypto agility of a security protocol design.

3.3. Cryptographic Key Establishment

Some environments will restrict the key-establishment approaches by policy. Such policies tend to improve interoperability within a particular environment, but they cause problems for individuals who need to work in multiple incompatible environments. In addition, administrators need to be aware that multiple environments are being used, track the policies, and enable the algorithms or cipher suites for each one of them.

Support for many key-establishment mechanisms in a security protocol offers more opportunity for crypto agility. Key establishment includes key-agreement mechanisms, key-transport mechanisms, and KEMs. Security protocol designers perform security analysis to ensure that all security goals are achieved when each of the possible key-establishment mechanisms is used.

Traditionally, security protocol designers have avoided support for more than one mechanism for exchanges that establish cryptographic keys because such support would make the security analysis of the overall protocol more difficult. When frameworks such as the Extensible Authentication Protocol (EAP) [24] are employed, the authentication mechanism often provides a session key in addition to providing authentication. As a result, key establishment is very flexible, but many of the cryptographic details are hidden from the application, which makes security analysis more difficult. Furthermore, this flexibility results in protocols that support multiple key-establishment mechanisms. In fact, the key-establishment mechanism itself is negotiable, which creates a design challenge to protect the negotiation of the key-establishment mechanism before it is used to produce cryptographic keys.

When security protocols support a single key-establishment mechanism, the security analysis is much more straightforward; however, crypto agility is reduced.

3.4. Balancing Security Strength and Protocol Complexity

When specifying a cipher suite, the relative strength of each algorithm should be roughly equal. Complexity in security protocols needs to be avoided. Each of these design goals is explored further in this section.

3.4.1. Balancing the Security Strength of Algorithms in a Cipher Suite

When selecting or negotiating a cipher suite, the relative strength of each algorithm needs to be considered. The algorithms in a cipher suite ought to provide roughly equal security strengths. The security protections provided by each algorithm in a particular context need to be considered when making the selection. Algorithm strength needs to be considered at the time a security protocol is designed, implemented, deployed, and configured. Advice from experts about relative algorithm strengths is useful, but in reality, such advice is often unavailable to system administrators who are deploying a protocol implementation. For this reason, SDOs should provide clear guidance to implementers, leading to balanced options being available at the time of deployment.

Performance is always a factor in selecting cryptographic algorithms. Performance and security need to be balanced. Users will not employ security features if the application runs too slowly when they are used. Some algorithms offer flexibility in their strength by adjusting the key size, number of rounds, authentication tag size, prime group size, and so on. For example, AES-128 is more efficient than AES-256, but it also offers less security.

3.4.2. Balancing Protocol Complexity

Security protocol design complexity leads to implementation complexity, which in turn makes vulnerabilities more likely. Thus, complexity should be avoided. Optional features can add complexity. Streamlining security protocols reduces less-used parts of the implementation. A security protocol with fewer options means there is a lower burden on implementation testing and a decreased attack surface, which makes it harder for attackers to discover vulnerabilities.

Security protocol designs need to anticipate changes to the supported set of cryptographic algorithms over time. Security protocol implementations avoid complexity to reduce vulnerability to attacks. For example, complex algorithm or cipher suite negotiation provides opportunities for downgrade attacks. Support for many algorithm alternatives is also harmful because of the challenges in deciding which algorithms are acceptable in a particular environment and maintaining that list of algorithms over time.

Protocol complexity can lead to portions of the implementation that are rarely used, increasing the opportunity for undiscovered, exploitable implementation bugs.

4. Crypto Agility for Applications

A cryptographic application programming interface (crypto API) separates the implementation of applications that make use of the cryptographic algorithms (e.g., email and web apps) from the implementation of the cryptographic algorithms themselves. This separation allows the application to focus on the high-level, application-specific details, while the cryptographic algorithms are implemented by a provider or a library to handle symmetric encryption, digital signature generation and verification, hashing, random number generation, key establishment, and so on.

For example, crypto APIs separate AES-CCM [25] and AES-GCM [26], which are both authenticated encryption with associated data (AEAD) algorithms, from application implementations by allowing an application to make the same crypto API calls to use either algorithm. Careful selection of default parameter values in the crypto API can make the interface to these two algorithms essentially identical, which facilitates future transition to a new AEAD algorithm.

Some crypto APIs offer implementations of security protocols like TLS or IPsec to further unburden the application. These protocol implementations depend on the crypto API for cryptographic operations. The application provides the list of algorithms or cipher suites that are available and acceptable, and then the algorithm negotiation capabilities for the protocol determine the algorithms that are actually used in the protocol.

To achieve crypto agility, system designers must introduce mechanisms that streamline the replacement of cryptographic algorithms in software, hardware, and infrastructures. These mechanisms will, at the same time, increase complexity. Therefore, system designers must make sure that the cryptographic interface is easy to use and well documented in order to reduce the risk of errors. Additionally, clear guidance must be provided for practitioners to follow.

4.1. Using an API in a Crypto Library Application

A cryptographic service provider (CSP) is an implementation of one or more cryptographic algorithms that is accessible by applications through a crypto API; see Fig. 2. CSPs are sometimes associated with protected key storage. For example, a CSP associated with a Trusted Platform Module (TPM) will also provide access to the asymmetric private keys that are stored on the TPM.

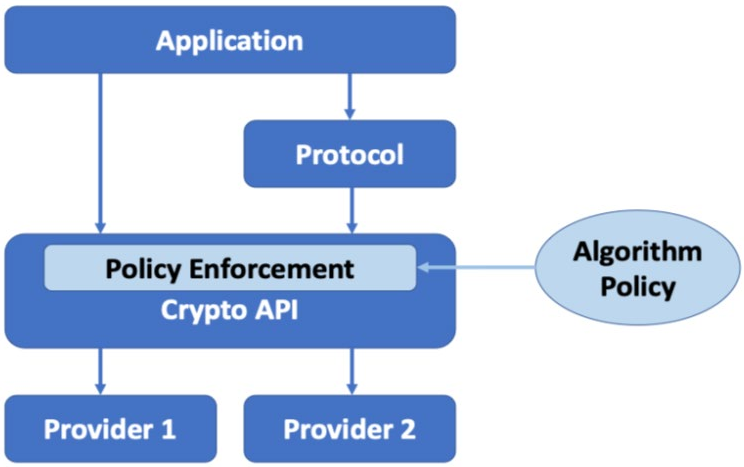


Fig. 2. Applications using crypto APIs

Cryptographic algorithm policy is set by the system administrator, which might be done to implement policy set by the enterprise Chief Information Security Officer (CISO). The policy will indicate whether a particular algorithm is allowed. For example, if there is a provider for Triple DES, calls to encrypt with it will fail if policy does not allow Triple DES. However, calls for Triple DES decryption might still be allowed so that stored files or email messages can be decrypted.

Some protocols are implemented in user space, an area in memory where applications execute that is distinct from kernel space. Kernel space is a part of a computer's memory where the operating system runs. For example, application-chosen TLS crypto library applications operate in user space; in fact, most libraries like OpenSSL, BoringSSL, Bouncy Castle, Network Security Services (NSS), and OpenSSH run in user space. Application developers need to consider whether the API is provided via the command line interface (CLI) or by "compiling in" support.

For software libraries, it is important to facilitate efficient updates. Some standard mechanisms must be in place to avoid security pitfalls in library updates.

4.2. Using APIs in the Operating System Kernel

Some security protocols run in the operating system kernel, a computer program that generally is loaded first when the system is turned on and has complete control over all system resources accessible to all application programs in the system. For example, in the case of IPsec, the datagram encryption and authentication provided by IPsec need to operate in the kernel. Similarly, disk encryption needs to run in the kernel.

To provide crypto agility in this case, the crypto API must also be accessible within the kernel. In some operating systems, only a subset of the crypto API's overall capabilities is available in the kernel. This subset is determined by the cryptographic operations required in the kernel. In many operating systems, the supported algorithms in the kernel are established when the kernel is built, meaning that plugins to add algorithms are not available in the kernel.

Some systems perform self-tests of the cryptographic functions as part of the operating system boot process. These tests ensure that the cryptographic operations are working as expected before the system is available to applications or users.

4.3. Hardware

There are several aspects of the hardware implementation of cryptographic algorithms to consider that are related to crypto agility.

A whole chip might be dedicated to the implementation of one cryptographic algorithm, or a small portion of a chip might implement a particular building-block function in support of a single cryptographic algorithm. In either case, a low-level interface is needed that works well in a particular hardware environment. In most cases, firmware is needed to manage memory and invoke the various low-level functions in the proper order. The functions that are implemented in the integrated circuit cannot be changed; this makes them well protected from attackers, but it also means that the chip will need to be replaced if it has design errors or changes are needed for the algorithms to be used.

Some chips, like Subscriber Identity Module (SIM) cards and TPMs, are dedicated to cryptographic operations. These chips are part of a larger computer system like a mobile phone or a laptop computer. These chips store the private keys and perform cryptographic operations that depend on the keys. At no time does the private keying material leave the chip. These chips support very few cryptographic algorithms, and changing algorithms is accomplished by replacing the chip. In fact, some devices offer a slot to do so without opening the device.

Hardware security modules (HSMs) are special-purpose hardware devices that store the private keys and perform cryptographic operations using those keys. An HSM might be a rack-mounted device for an organization or high-value application, or it might be a portable device that is easily locked in a safe when not in use. At no time does the private keying material leave the HSM, but there are operations to securely back up the private keying material to another HSM. Note that HSMs provide cryptographic services, but they also consume cryptographic services. HSMs offer tamper-detection capabilities to protect the private keying material stored in them. HSMs often include a microprocessor as well as one or more chips that are designed to accelerate different cryptographic algorithms or parts of the algorithms invoked by software cryptographic implementations.

A personal portable cryptographic token, such as a Personal Identity Verification (PIV) card or a USB token, is a device that stores the private keys for an individual. The human user plugs the portable device into whatever computer they are using. At no time does the keying material leave the portable device. These devices are essentially tiny HSMs intended to be used by one person.

Some central processing units (CPUs) have instructions that were designed to accelerate specific algorithms. A cryptographic algorithm implementation might detect whether such instructions are available and then take advantage of them if they are. For example, the Intel SHA Extensions paper [27] states that the CPUs offer features to make SHA hash computations faster.

From this discussion, it should be clear that there are many reasons an application might use hardware to support cryptographic operations, including performance, the protection of private keys, and portability. An additional reason is that some hardware offers a good source of random numbers, which are vital to the generation of quality keying material.

On the other hand, it is easier to provide multiple cryptographic algorithms to facilitate agility in library and application software than in hardware. Once a chip leaves the factory, additional algorithms may not be added easily. Other layers in an architecture fall on a spectrum between these two cases. The crypto API needs to be designed so that all points on this spectrum are accommodated. In some environments, especially HSMs and other cryptographic tokens, the data needs to move to the device where the key is stored for the data to be protected using that key.

For the environments where the update of cryptographic functions in hardware is not possible in the field, it is important to consider the use of state-of-the-art cryptography to include implementations of the best and most conservative variants for each cryptographic function. A key element is the communication between cryptographers and developers to decide on a long-term plan based on the best estimate of the security needs during the lifetime of a specific hardware device. For example, secure booting (i.e., starting a computer and loading its operating system) requires using digital signature schemes. The public key and the program for verifying the signatures are included in the boot code and cannot be updated. In this case, to make sure that the platform is trustable during its lifetime, the signature schemes must be able to provide the required security during the lifetime of the device.

5. Discussions

Achieving crypto agility demands collaborations and communications among cryptographers, developers, implementers, and practitioners to manage the risk of using cryptography to secure the data. To be actionable, crypto agility requirements must be specific for each implementation and application environment. This section discusses tradeoffs and identifies some areas for future work. Each subsection highlights important areas for consideration and associated stakeholders.

5.1. Resource Considerations

Resource limitation is the most difficult challenge to deal with for achieving crypto agility. This section discusses resource considerations for protocol designers, hardware implementers, and cryptographers.

Crypto agility requires support for multiple cryptographic algorithms in a protocol. Some algorithms have much larger public keys, signatures, or ciphertext than the algorithms being replaced. Experience has shown that large sizes challenge the limits of existing protocols. It is important for protocol designers to consider resource demands in order to plan for future transitions and to distinguish intrinsic limitations from shortsighted design decisions.

Hardware implementation is limited by capacity. It may not be possible to implement many algorithms in one hardware platform. Some optimization efforts such as accelerator reuse have been considered. Further research is needed in this area to deal with the transition from traditional public-key cryptography to post-quantum cryptography.

Future cryptographic algorithm design must consider resource limitations. Usually, each design has focused on the resource requirements of a single algorithm for an application without considering other applications. For example, the design may use a specific primitive or a subroutine (such as a hash function) that is not commonly used by other algorithms. To save hardware resources, it is desirable for different algorithms to share the same subroutines. Cryptographers have considered algorithms based on diversified assumptions so that when one assumption is determined to be incorrect, an alternative based on a different assumption is in place. Achieving crypto agility within resource limitations requires cryptographers to prioritize security-related diversities. This is a new area of research that must take a different approach from that of traditional approaches using a single algorithm design.

5.2. Agility-Aware Design

This section discusses agility design considerations for application, platform, and protocol designers.

Current practice has made it possible for applications to access cryptographic services through APIs. This significantly eases a cryptographic transition from one algorithm to another. When an algorithm is found vulnerable and must no longer be used, the use of an API can enable the transition by providing a flexible and efficient way to manage cryptographic operations.

However, for some operating systems, cryptographic operations are determined at the time when the kernels are built. In this case, it is not possible to update the cryptographic operations in the kernel when a transition is needed because they are an integral part of the kernel. A working area to be considered is to improve API usage in kernels to support update and transition.

Agility-aware design could be reflected in the product or system configuration. It would ensure that the user interface (UI) and API can support new algorithms with different key and parameter sizes in order to use the underlying cryptographic software libraries and hardware accelerators. The design would not make assumptions based on one algorithm or a family of algorithms when coding cryptographic implementations. That is, the design would ensure that buffers, memory locations, storage, etc. could handle large keys and parameters.

Some well-deployed security protocols, such as TLS, facilitate authenticated cipher-suite negotiation to allow adding new algorithms to and discontinuing the use of weak algorithms from the available cipher suites. This should be a common practice in any protocol design. For example, in most of the IETF Requests for Comment (RFCs), there is a section called “Security considerations.” It may be beneficial to include a section about “Crypto agility considerations” in the standards to provide rationales for the design choices to allow crypto agility.

5.3. Complexity and Security

Accommodating crypto agility introduces complexity to protocols and systems that protocol designers and system architects and implementers should take into consideration. It can also increase attack surfaces. For example, if cipher suite negotiation integrity is not properly protected, a downgrade attack can lead to a weaker cipher suite than otherwise agreed upon. For software libraries and APIs, a larger number of options may increase the chance to introduce security bugs. For enterprise IT administrators, it is important to make sure that the configuration is updated to reflect new security requirements.

Crypto agility requires sound mechanisms to assure a secure and smooth transition. Currently, most security analysis and evaluations focus on a protocol or a system configuration without considering transition mechanisms. Cryptographic transition mechanisms should be included in a security assessment for a protocol or a system configuration.

5.4. Crypto Agility in the Cloud

This section discusses agility consideration for cloud computing service architects, developers, operators, and cryptographers.

The cloud refers to many remote servers accessible over the internet where users can store data, run applications, and access services. Cloud service providers need crypto agility to meet the diverse requirements of various customers. Some applications use a cloud to support cryptographic operations. A cloud environment is advantageous for several reasons, including scalability.

However, use of a cloud environment tends to lock developers into a particular crypto API. For example, a cloud environment might provide benefits of secure backup and affordable cloud-hosting services using container technology to ensure that a software package and all its dependencies run quickly and reliably. Based on the choices made by the cloud provider, the developers are also locked into the hardware, potentially including HSM support.

In contrast, some cloud providers offer the ability to access an application-specific HSM that is external to the cloud environment. This avoids provider lock-in, but it comes with many operational requirements for the application provider to ensure availability. Additional choices, such as selecting the confidential computing architecture to protect data that is processed, can prevent the cloud provider from accessing keying material, but ultimately the cloud provider can remove the entire application. In some cloud environments, the cloud provider may be able to administratively delete keys from an HSM, even if they are not able to otherwise access those keys.

5.5. Maturity Assessment for Crypto Agility

This section introduces the consideration for all stakeholders in the organization to have a crypto agility maturity model to measure and track the maturity of the state of crypto agility against industry standards and best practices in order to be resilient against the evolving changes in crypto requirements.

A maturity model is needed for a given software or IT landscape to assess the readiness of a software or system for cryptographic algorithm transition. Hohm, Heinemann, and Wiesmaier [28] proposed a Crypto Agility Maturity Model (CAMM), which identifies five maturity levels from level 0 to level 4. These five levels are described as not possible, possible, prepared, practiced, and sophisticated. The requirements associated with each of the five levels make it possible to evaluate a given system according to its ability to implement crypto-agility requirements. The requirements in [28] are categorized as knowledge, process, and system properties. These requirements are valuable references in considering the maturity of crypto agility. For example, at Level 2-Prepared, Requirement 2.0 Cryptographic Modularity “is understood as a system design that enables changes to the cryptographic components without affecting the functionality of the other system components. In the event of a vulnerability, the implementation of cryptographic functions, their parameters and primitives can be replaced without affecting the system logic.” This requirement is noted as system properties. However, most of the requirements are descriptive, not quantitative. Some of the requirements apply to a system, while the others may apply to an organization or a set of protocols.

This work could be adapted to create a concrete assessment model that is applicable to every system. If this happens, the government can create incentives for the adoption of such a maturity model. Maturity assessment for crypto agility is a new area to explore. FIPS 140 validation does not currently assess crypto agility, but the feasibility of adapting FIPS 140 testing to encompass crypto agility is being studied.

Current FIPS 140 validation tests the implementations of NIST-approved cryptographic algorithms. The resulting certification includes a list of NIST-approved algorithms implemented

in a cryptographic module. If a module implements multiple algorithms at multiple security strengths for the same function (for example, the SHA-2 and SHA-3 hash functions), then the module supports crypto agility for hash functions. However, if the only implemented algorithm will be deprecated in the next few years (ECDSA, for example, with only elliptic curve P-224), this raises an issue about a module's support for crypto agility.

5.6. Crypto Agility Strategic Plan for Managing Organizations' Crypto Risks

A crypto agility strategic plan as presented in Fig. 3 brings together key functions such as governance, crypto and data assets, risk management, and automated tooling to inform the migration/transition of crypto at different technology levels. Organizations need to transition or migrate their cryptographic use multiple times throughout the systems' lifetimes. By incorporating crypto agility into their crypto policies during technology refreshes, updates, or modernization efforts, organizations can proactively address emerging threats, technological advances, system weaknesses, and evolving business requirements, standards, regulations, and mandates.

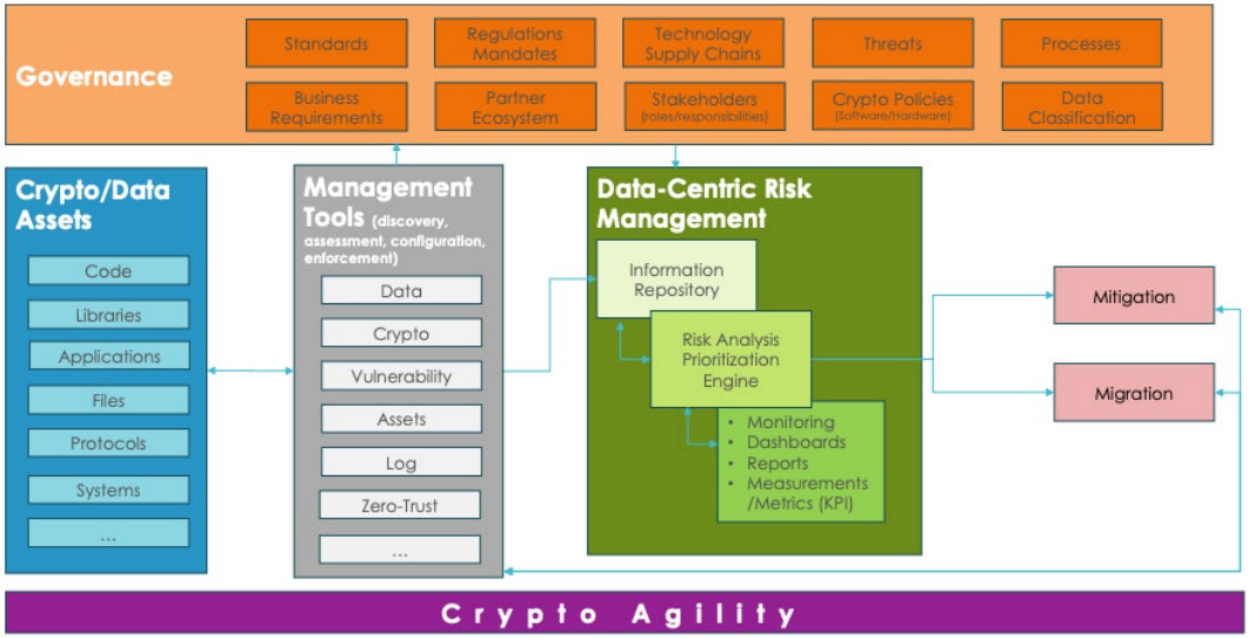


Fig. 3. Crypto agility strategic plan for managing organizations' crypto risks

The plan may include several key activities, including:

- Integrate crypto agility into the organization's existing governance function to establish, communicate, and monitor the cybersecurity risk management strategy, expectations, and policies related to cryptography. This includes understanding crypto standards, regulations, and mandates, and communicating these requirements to data owners, IT and development teams, business partners, and technology supply chain vendors prioritized by the criticality of the data.

- Inventory the use of cryptography for data protection across the organization by adopting a data-centric approach informed by the criticality of the data to identify the organization's most valuable assets, such as application codes, libraries, software, hardware, user-generated content, communication protocols, enterprise services, and systems.
- Identify gaps in enterprise management tools for managing assets, configurations, vulnerabilities, and logs. These tools should support crypto risk management and data protection functions by automating the identification, assessment, characterization, enforcement, and monitoring of crypto use across the assets in an automated way. If necessary, enhance the tools with automated data and cryptographic discovery capabilities, including algorithms and key lengths. For instance, vulnerability management and software/hardware development tools can help ensure comprehensive visibility and an inventory of assets such as code, libraries, applications, and associated cryptographic algorithms.
- Develop a prioritization list of assets to be mitigated first due to the use of weak cryptography, based on the disparate data collected from the initial steps. A crypto policy-informed risk assessment engine analyzes this data to form a strategy and recommend actions to reduce risks. The engine continuously measures, monitors, and reports on the state of crypto, particularly focusing on crypto agility key performance indicators (KPIs) for the level of efforts to adapt and migrate effectively and efficiently, based on the organization's defined crypto policy.
- Implement the strategy and actions based on the prioritization list. Crypto agility is crucial for deciding whether to migrate assets smoothly or deploy mitigation techniques to reduce risks. Organizations can use enterprise management tools to migrate assets, such as code, applications, software, hardware, and communication protocols, or implement additional security controls as part of a zero-trust approach [29] to mitigate crypto risks for networks, devices, and applications if the assets are not agile enough to support the crypto policy.

These steps are continuously repeated to mitigate evolving crypto risks and enhance the crypto agility posture within organizations. Crypto agility is a key principle that organizations should consider throughout the data-centric cryptographic risk management process.

Crypto governance is an important part of a crypto agility strategic plan. The following subsections discuss some components of governance that are crucial for organizations to drive cryptographic practices and compliance in support of managing the crypto risks among all stakeholders, from the organization's board to the implementers.

5.6.1. Crypto Standards, Regulations, and Mandates

Any crypto agility effort must consider the effects of standards, regulations, and mandates on transition requirements for cryptographic algorithms. Movements to achieve crypto agility involve coordination between protocol designers, software and hardware vendors, application

and standards developers, policy makers, and IT administrators. Government standards and regulations can mandate the transition when an algorithm is found vulnerable. NIST SP 800-131A guides algorithm and security strength transitions by setting transition schedules for implementers to sunset certain algorithms or security strengths based on a common understanding of the computing power available for attackers and the latest research results. For example, SP 800-131A rev. 2 [30], published in 2019, set the end of 2023 as the date to disallow three-key Triple DES for applying cryptographic protection.

Industry standards play an important role in compliance with security requirements for cryptographic algorithm usage in different application environments. The standards for different applications such as internet protocols, communications, and applications update the supported cipher suites to eliminate algorithms and ciphers that are vulnerable. Security protocols often define mandatory-to-implement cipher suites to reflect the state-of-the-art of cryptography and support interoperability.

The NIST Cryptographic Algorithm Validation Program (CAVP) provides validation testing for FIPS-approved and NIST-recommended cryptographic algorithms. Cryptographic algorithm validation is a prerequisite of cryptographic module validation. The approved algorithms and relevant parameter sets are updated based on transition requirements.

From a practitioner's perspective, certain policies, laws, and mechanisms must be established to enhance crypto agility practice to facilitate the transition and provide proper security during the transition. These laws and policies are coupled with industry-specific requirements. It is very important to handle the data in a secure way during a transition. For example, for the encrypted storage of data-at-rest, a mechanism must be established to handle encrypted user data when the encryption algorithm is to be replaced by a stronger one. Similarly, when a digital signature algorithm must be replaced, a mechanism to handle already-signed documents is required.

5.6.2. Crypto Security Policy Enforcement

Crypto security policy enforcement must be considered as an important factor in the crypto agility assessment for each protocol, system, and application. One of the most challenging aspects of crypto agility is replacing vulnerable algorithms in a timely manner and at the same time keeping the system running without interruption. For security protocols, a crypto security policy can be enforced through specifying mandatory-to-implement algorithms and disallowing the use of weak algorithms in a timely fashion. For a system, a security policy can be enforced through the use of an API. Security practitioners enforce security policy through decisions for using cryptographic algorithms with required security strengths.

Enforcing crypto security policy requires communications among cryptographers, developers, practitioners, IT administrators, and policy makers. Each decision on deprecating a cryptographic algorithm must be synchronized among all the stakeholders so the security policy can be updated quickly.

6. Conclusion

Crypto agility is a future-proofing strategy to deal with changes. It demands communications among cryptographers, developers, implementers, and practitioners to accommodate evolving security, performance, and interoperability challenges. The pursuit of crypto agility capabilities involves exploration of new technologies and management schemes. New crypto agility requirements must be developed for each environment. The security analysis and evaluation for protocols, systems, and applications must include mechanisms for transitions.

References

[1] National Academies of Sciences, Engineering, and Medicine (2016) Cryptographic Agility and Interoperability: Proceedings of a Workshop. Forum on Cyber Resilience Workshop Series. (The National Academies Press, Washington, DC). https://doi.org/10.17226/24636

[2] National Institute of Standards and Technology (1999) Data Encryption Standard (DES). (U.S. Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) 46-3. Withdrawn May 19, 2005. Available at https://csrc.nist.gov/pubs/fips/46-3/final

[3] National Institute of Standards and Technology (2001) Advanced Encryption Standard (AES). (U.S. Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) 197-upd1, updated May 9, 2023. https://doi.org/10.6028/NIST.FIPS.197-upd1

[4] National Institute of Standards and Technology (2015) Secure Hash Standard (SHS). (U.S. Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) 180-4. https://doi.org/10.6028/NIST.FIPS.180-4

[5] Wang X, Yin YL, Yu H (2005) Finding Collisions in the Full SHA-1. Advances in Cryptology — CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621. (Springer, Berlin, Heidelberg). https://doi.org/10.1007/11535218_2

[6] Rescorla E (2018) The Transport Layer Security (TLS) Protocol Version 1.3. (Internet Engineering Task Force (IETF)), IETF Request for Comments (RFC) 8446. https://doi.org/10.17487/RFC8446

[7] Leurent G, Peyrin T (2020). SHA-1 is a Shambles - First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust. SEC'20: Proceedings of the 29th USENIX Conference on Security Symposium. Available at https://eprint.iacr.org/2020/014

[8] National Institute of Standards and Technology (2022) NIST Transitioning Away from SHA-1 for All Applications. Available at https://csrc.nist.gov/news/2022/nist-transitioning-away-from-sha-1-for-all-apps

[9] National Institute of Standards and Technology (2000) Digital Signature Standard (DSS). (U.S. Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) 186-2. Withdrawn October 5, 2001. Available at https://csrc.nist.gov/pubs/fips/186-2/final

[10] Barker E (2020) Recommendation for Key Management: Part 1 – General. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-57 Part 1, Revision 5. https://doi.org/10.6028/NIST.SP.800-57pt1r5

[11] Moody D, Perlner R, Regenscheid A, Robinson A, Cooper D (2024) Transition to Post-Quantum Cryptography Standards. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Internal Report (IR) 8547. https://doi.org/10.6028/NIST.IR.8547.ipd

[12] National Institute of Standards and Technology (2024) Module-Lattice-Based Digital Signature Standard. (U.S. Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) 204. https://doi.org/10.6028/NIST.FIPS.204

[13] Dierks T, Rescorla E (2008) The Transport Layer Security (TLS) Protocol Version 1.2. (Internet Engineering Task Force (IETF)), IETF Request for Comments (RFC) 5246. https://doi.org/10.17487/RFC5246
[14] Kaufman C, Hoffman P, Nir Y, Eronen P, Kivinen T (2014) Internet Key Exchange Protocol Version 2 (IKEv2). Internet Engineering Task Force (IETF). Request for Comments (RFC) 72966. https://doi.org/10.17487/RFC7296
[15] Ramsdell B (2004) Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. (Internet Engineering Task Force (IETF)), IETF Request for Comments (RFC) 3851. https://doi.org/10.17487/RFC3851
[16] Ramsdell B, Turner S (2010) Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2. (Internet Engineering Task Force (IETF)), IETF Request for Comments (RFC) 5751. https://doi.org/10.17487/RFC5751
[17] Housley R (2009) Cryptographic Message Syntax (CMS). (Internet Engineering Task Force (IETF)), IETF Request for Comments (RFC) 5652. https://doi.org/10.17487/RFC5652
[18] Crocker S, Rose S (2013) Signaling Cryptographic Algorithm Understanding in DNS Security Extensions (DNSSEC). (Internet Engineering Task Force (IETF)), IETF Request for Comments (RFC) 6975. https://doi.org/10.17487/RFC6975
[19] Gagliano R, Kent S, Turner S (2013) Algorithm Agility Procedure for the Resource Public Key Infrastructure (RPKI). (Internet Engineering Task Force (IETF)), IETF Request for Comments (RFC) 6916. https://doi.org/10.17487/RFC6916
[20] Popov A (2015) Prohibiting RC4 Cipher Suites. (Internet Engineering Task Force (IETF)), IETF Request for Comments (RFC) 7465. https://doi.org/10.17487/RFC7465
[21] Cooper D, Santesson S, Farrell S, Boeyen S, Housley R, Polk W (2008) Internet X.509 Public Key Infrastructure Certification and Certificate Revocation List (CRL) Profile. (Internet Engineering Task Force (IETF)), IETF Request for Comments (RFC) 5280. https://doi.org/10.17487/RFC5280
[22] Barker EB, Chen L, Roginsky AL, Vassilev A, Davis R (2018) Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-56A, Rev. 3. https://doi.org/10.6028/NIST.SP.800-56Ar3
[23] National Institute of Standards and Technology (2024) Module-Lattice-Based Key-Encapsulation Mechanism Standard. (U.S. Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) 203. https://doi.org/10.6028/NIST.FIPS.203
[24] Aboba B, Blunk L, Vollbrecht J, Carlson J, Levkowetz H (2004) Extensible Authentication Protocol (EAP). (Internet Engineering Task Force (IETF)), IETF Request for Comments (RFC) 3748. https://doi.org/10.17487/RFC3748
[25] Dworkin MJ (2004) Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-38C, Includes updates as of July 20, 2007. https://doi.org/10.6028/NIST.SP.800-38C
[26] Dworkin MJ (2007) Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-38D. https://doi.org/10.6028/NIST.SP.800-38D

[27] Gulley S, Gopal V, Yap K, Feghali W, Guilford J, Wolrich G (2013) Intel SHA Extensions: New Instructions Supporting the Secure Hash Algorithm on Intel Architecture Processors. (Intel Corporation.) Available at https://www.intel.com/content/dam/develop/external/us/en/documents/intel-sha-extensions-white-paper-402097.pdf

[28] Hohm J, Heinemann A, Wiesmaier A (2022) Towards a Maturity Model for Crypto-Agility Assessment. Available at https://arxiv.org/abs/2202.07645

[29] Rose SW, Borchert O, Mitchell S, Connelly S (2020) Zero Trust Architecture. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-207. https://doi.org/10.6028/NIST.SP.800-207

[30] Barker EB, Roginsky AL (2019) Transitioning the Use of Cryptographic Algorithms and Key Lengths. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-131A, Rev. 2. https://doi.org/10.6028/NIST.SP.800-131Ar2

Appendix A. List of Symbols, Abbreviations, and Acronyms

AEAD
Authenticated Encryption with Associated Data

AES
Advanced Encryption Standard

AES-CCM
Advanced Encryption Standard – Counter with CBC-MAC

AES-GCM
Advanced Encryption Standard – Galois/Counter Mode

API
Application Programming Interface

CA
Certification Authority

CAMM
Crypto Agility Maturity Model

CAVP
Cryptographic Algorithm Validation Program

CISO
Chief Information Security Officer

CLI
Command Line Interface

CMS
Cryptographic Message Syntax

CPU
Central Processing Unit

CRQC
Cryptographically Relevant Quantum Computer

CSP
Cryptographic Service Provider

DES
Data Encryption Standard

DNSSEC
Domain Name System Security Extensions

EAP
Extensible Authentication Protocol

ECDSA
Elliptic Curve Digital Signature Algorithm

978 **EDNS**
979 Extension Mechanisms for Domain Name System

980 **FIPS**
981 Federal Information Processing Standard

982 **HSM**
983 Hardware Security Module

984 **IETF**
985 Internet Engineering Task Force

986 **IKE**
987 Internet Key Exchange

988 **IPsec**
989 Internet Protocol Security

990 **IR**
991 Internal Report

992 **KEM**
993 Key Encapsulation Mechanism

994 **KPI**
995 Key Performance Indicator

996 **MAC**
997 Message Authentication Code

998 **ML-DSA**
999 Module-Lattice-Based Digital Signature Algorithm

1000 **ML-KEM**
1001 Module-Lattice-Based Key Encapsulation Mechanism

1002 **PIV**
1003 Personal Identity Verification

1004 **PKI**
1005 Public Key Infrastructure

1006 **PQC**
1007 Post-Quantum Cryptography

1008 **RFC**
1009 Request for Comment

1010 **RPKI**
1011 Resource Public Key Infrastructure

1012 **RSA**
1013 Rivest-Shamir-Adelman

1014 **SDO**
1015 Standards Developing Organization

SHA
Secure Hash Algorithm

SIM
Subscriber Identity Module

S/MIME
Secure Multipurpose Internet Mail Extensions

SP
Special Publication

TLS
Transport Layer Security

TPM
Trusted Platform Module

UI
User Interface

USB
Universal Serial Bus