CWE-1240: Use of a Cryptographic Primitive with a Risky Implementation

Weakness ID: 1240
Vulnerability Mapping: ALLOWED
Abstraction: Base

View customized information: Conceptual | Operational | Mapping Friendly | Complete | Custom

Description

To fulfill the need for a cryptographic primitive, the product implements a cryptographic algorithm using a non-standard, unproven, or disallowed/non-compliant cryptographic implementation.

Extended Description

Cryptographic protocols and systems depend on cryptographic primitives (and associated algorithms) as their basic building blocks. Some common examples of primitives are digital signatures, one-way hash functions, ciphers, and public key cryptography; however, the notion of "primitive" can vary depending on point of view. See "Terminology Notes" for further explanation of some concepts.

Cryptographic primitives are defined to accomplish one very specific task in a precisely defined and mathematically reliable fashion. For example, suppose that for a specific cryptographic primitive (such as an encryption routine), the consensus is that the primitive can only be broken after trying out N different inputs (where the larger the value of N, the stronger the cryptography). For an encryption scheme like AES-256, one would expect N to be so large as to be infeasible to execute in a reasonable amount of time.

If a vulnerability is ever found that shows that one can break a cryptographic primitive in significantly less than the expected number of attempts, then that primitive is considered weakened (or sometimes in extreme cases, colloquially it is "broken"). As a result, anything using this cryptographic primitive would now be considered insecure or risky. Thus, even breaking or weakening a seemingly small cryptographic primitive has the potential to render the whole system vulnerable, due to its reliance on the primitive. A historical example can be found in TLS when using DES. One would colloquially call DES the cryptographic primitive for transport encryption in this version of TLS. In the past, DES was considered strong, because no weaknesses were found in it; importantly, DES has a key length of 56 bits. Trying N=2^56 keys was considered impractical for most actors. Unfortunately, attacking a system with 56-bit keys is now practical via brute force, which makes defeating DES encryption practical. It is now practical for an adversary to read any information sent under this version of TLS and use this information to attack the system. As a result, it can be claimed that this use of TLS is weak, and that any system depending on TLS with DES could potentially render the entire system vulnerable to attack.

Cryptographic primitives and associated algorithms are only considered safe after extensive research and review from experienced cryptographers from academia, industry, and government entities looking for any possible flaws. Furthermore, cryptographic primitives and associated algorithms are frequently reevaluated for safety when new mathematical and attack techniques are discovered. As a result and over time, even well-known cryptographic primitives can lose their compliance status with the discovery of novel attacks that might either defeat the algorithm or reduce its robustness significantly.

If ad-hoc cryptographic primitives are implemented, it is almost certain that the implementation will be vulnerable to attacks that are well understood by cryptographers, resulting in the exposure of sensitive information and other consequences.

This weakness is even more difficult to manage for hardware-implemented deployment of cryptographic algorithms. First, because hardware is not patchable as easily as software, any flaw discovered after release and production typically cannot be fixed without a recall of the product. Secondly, the hardware product is often expected to work for years, during which time computation power available to the attacker only increases. Therefore, for hardware implementations of cryptographic primitives, it is absolutely essential that only strong, proven cryptographic primitives are used.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	**Technical Impact:** *Read Application Data* Incorrect usage of crypto primitives could render the supposedly encrypted data as unencrypted plaintext in the worst case.	High

Potential Mitigations

Phase: Requirements

Require compliance with the strongest-available recommendations from trusted parties, and require that compliance must be kept up-to-date, since recommendations evolve over time. For example, US government systems require FIPS 140-3 certification, which supersedes FIPS 140-2 [REF-1192] [REF-1226].

Effectiveness: High

Phase: Architecture and Design

Ensure that the architecture/design uses the strongest-available primitives and algorithms from trusted parties. For example, US government systems require FIPS 140-3 certification, which supersedes FIPS 140-2 [REF-1192] [REF-1226].

Effectiveness: High

Phase: Architecture and Design

Do not develop custom or private cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. As with all cryptographic mechanisms, the source code should be available for analysis. If the algorithm may be compromised when attackers find out how it works, then it is especially weak.

Effectiveness: Discouraged Common Practice

Phase: Architecture and Design

Try not to use cryptographic algorithms in novel ways or with new modes of operation even when you "know" it is secure. For example, using SHA-2 chaining to create a 1-time pad for encryption might sound like a good idea, but one should not do this.

Effectiveness: Discouraged Common Practice

Phase: Architecture and Design

Ensure that the design can replace one cryptographic primitive or algorithm with another in the next generation ("cryptographic agility"). Where possible, use wrappers to make the interfaces uniform. This will make it easier to upgrade to stronger algorithms. This is especially important for hardware, which can be more difficult to upgrade quickly than software; design the hardware at a replaceable block level.

Effectiveness: Defense in Depth

Phase: Architecture and Design

Do not use outdated or non-compliant cryptography algorithms. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong [REF-267].

Effectiveness: Discouraged Common Practice

Phases: Architecture and Design; Implementation

Do not use a linear-feedback shift register (LFSR) or other legacy methods as a substitute for an accepted and standard Random Number Generator.

Effectiveness: Discouraged Common Practice

Phases: Architecture and Design; Implementation

Do not use a checksum as a substitute for a cryptographically generated hash.

Effectiveness: Discouraged Common Practice

Phase: Architecture and Design

Strategy: Libraries or Frameworks

Use a vetted cryptographic library or framework. Industry-standard implementations will save development time and are more likely to avoid errors that can occur during implementation of cryptographic algorithms. However, the library/framework could be used incorrectly during implementation.

Effectiveness: High

Phases: Architecture and Design; Implementation

When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for the prevention of common attacks.

Effectiveness: Moderate

Phases: Architecture and Design; Implementation

Do not store keys in areas accessible to untrusted agents. Carefully manage and protect the cryptographic keys (see CWE-320). If the keys can be guessed or stolen, then the strength of the cryptography algorithm is irrelevant.

Effectiveness: Moderate

Relationships

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf		327	Use of a Broken or Risky Cryptographic Algorithm

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name
MemberOf	C	310	Cryptographic Issues

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name
MemberOf	C	1205	Security Primitives and Cryptography Issues

Modes Of Introduction

Phase	Note
Architecture and Design	This weakness is primarily introduced during the architecture and design phase as risky primitives are included.
Implementation	Even in cases where the Architectural phase properly specifies a cryptographically secure design, the design may be changed during implementation due to unforeseen constraints.

Applicable Platforms

Languages

Class: Not Language-Specific *(Undetermined Prevalence)*

Operating Systems

Class: Not OS-Specific *(Undetermined Prevalence)*

Architectures

Class: Not Architecture-Specific *(Undetermined Prevalence)*

Technologies

Class: System on Chip *(Undetermined Prevalence)*

Demonstrative Examples

Example 1

Re-using random values may compromise security.

(bad code)

```
Suppose an Encryption algorithm needs a random value for a key. Instead of using a DRNG (Deterministic Random Number Generator), the designer uses a linear-feedback shift register (LFSR) to generate the value.
```

While an LFSR may provide pseudo-random number generation service, the entropy (measure of randomness) of the resulting output may be less than that of an accepted DRNG (like that used in dev/urandom). Thus, using an LFSR weakens the strength of the cryptographic system, because it may be possible for an attacker to guess the LFSR output and subsequently the encryption key.

(good code)

```
If a cryptographic algorithm expects a random number as its input, provide one. Do not provide a pseudo-random value.
```

Observed Examples

Reference	Description
CVE-2020-4778	software uses MD5, which is less safe than the default SHA-256 used by related products
CVE-2005-2946	Default configuration of product uses MD5 instead of stronger algorithms that are available, simplifying forgery of certificates.
CVE-2019-3907	identity card uses MD5 hash of a salt and password
CVE-2021-34687	personal key is transmitted over the network using a substitution cipher
CVE-2020-14254	product does not disable TLS-RSA cipher suites, allowing decryption of traffic if TLS 2.0 and secure ciphers are not enabled.
CVE-2019-1543	SSL/TLS library generates 16-byte nonces but reduces them to 12 byte nonces for the ChaCha20-Poly1305 cipher, converting them in a way that violates the cipher's requirements for unique nonces.
CVE-2017-9267	LDAP interface allows use of weak ciphers
CVE-2017-7971	SCADA product allows "use of outdated cipher suites"
CVE-2020-6616	Chip implementing Bluetooth uses a low-entropy PRNG instead of a hardware RNG, allowing spoofing.
CVE-2019-1715	security product has insufficient entropy in the DRBG, allowing collisions and private key discovery
CVE-2014-4192	Dual_EC_DRBG implementation in RSA toolkit does not correctly handle certain byte requests, simplifying plaintext recovery
CVE-2007-6755	Recommendation for Dual_EC_DRBG algorithm contains point Q constants that could simplify decryption

Weakness Ordinalities

Ordinality	Description
Primary	*(where the weakness exists independent of other weaknesses)*

Detection Methods

Architecture or Design Review

Review requirements, documentation, and product design to ensure that primitives are consistent with the strongest-available recommendations from trusted parties. If the product appears to be using custom or proprietary implementations that have not had sufficient public review and approval, then this is a significant concern.

Effectiveness: High

Manual Analysis

Analyze the product to ensure that implementations for each primitive do not contain any known vulnerabilities and are not using any known-weak algorithms, including MD4, MD5, SHA1, DES, etc.

Effectiveness: Moderate

Dynamic Analysis with Manual Results Interpretation

For hardware, during the implementation (pre-Silicon / post-Silicon) phase, dynamic tests should be done to ensure that outputs from cryptographic routines are indeed working properly, such as test vectors provided by NIST [REF-1236].

Effectiveness: Moderate

Dynamic Analysis with Manual Results Interpretation

It needs to be determined if the output of a cryptographic primitive is lacking entropy, which is one clear sign that something went wrong with the crypto implementation. There exist many methods of measuring the entropy of a bytestream, from sophisticated ones (like calculating Shannon's entropy of a sequence of characters) to crude ones (by compressing it and comparing the size of the original bytestream vs. the compressed - a truly random byte stream should not be compressible and hence the uncompressed and compressed bytestreams should be nearly identical in size).

Effectiveness: Moderate

Memberships

Nature	Type	ID	Name
MemberOf	V	1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List
MemberOf	C	1402	Comprehensive Categorization: Encryption

Vulnerability Mapping Notes

Usage: ALLOWED *(this CWE ID could be used to map to real-world vulnerabilities)*

Reason: Acceptable-Use

Rationale:

This CWE entry is at the Base level of abstraction, which is a preferred level of abstraction for mapping to the root causes of vulnerabilities.

Comments:

Carefully read both the name and description to ensure that this mapping is an appropriate fit. Do not try to 'force' a mapping to a lower-level Base/Variant simply to comply with this preferred level of abstraction.

Notes

Terminology

Terminology for cryptography varies widely, from informal and colloquial to mathematically-defined, with different precision and formalism depending on whether the stakeholder is a developer, cryptologist, etc. Yet there is a need for CWE to be self-consistent while remaining understandable and acceptable to multiple audiences.

As of CWE 4.6, CWE terminology around "primitives" and "algorithms" is emerging as shown by the following example, subject to future consultation and agreement within the CWE and cryptography communities. Suppose one wishes to send encrypted data using a CLI tool such as OpenSSL. One might choose to use AES with a 256-bit key and require tamper protection (GCM mode, for instance). For compatibility's sake, one might also choose the ciphertext to be formatted to the PKCS#5 standard. In this case, the "cryptographic system" would be AES-256-GCM with PKCS#5 formatting. The "cryptographic function" would be AES-256 in the GCM mode of operation, and the "algorithm" would be AES. Colloquially, one would say that AES (and sometimes AES-256) is the "cryptographic primitive," because it is the algorithm that realizes the concept of symmetric encryption (without modes of operation or other protocol related modifications). In practice, developers and architects typically refer to base cryptographic algorithms (AES, SHA, etc.) as cryptographic primitives.

Maintenance

Since CWE 4.4, various cryptography-related entries, including CWE-327 and CWE-1240, have been slated for extensive research, analysis, and community consultation to define consistent terminology, improve relationships, and reduce overlap or duplication. As of CWE 4.6, this work is still ongoing.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-97	Cryptanalysis

References

[REF-267] Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001-05-25. <https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf>. *URL validated: 2023-04-07*.

[REF-1227] Wikipedia. "Cryptographic primitive". <https://en.wikipedia.org/wiki/Cryptographic_primitive>.

[REF-1226] Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-2: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001-05-25. <https://csrc.nist.gov/publications/detail/fips/140/2/final>.

[REF-1192] Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-3: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2019-03-22. <https://csrc.nist.gov/publications/detail/fips/140/3/final>.

[REF-1236] NIST. "CAVP Testing: Individual Component Testing". Test Vectors. <https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/component-testing>.

Content History

Submissions

Submission Date	Submitter	Organization
2020-02-10 *(CWE 4.0, 2020-02-24)*	Arun Kanuparthi, Hareesh Khattri, Parbati Kumar Manna, Narasimha Kumar V Mangipudi	Intel Corporation

Contributions

Contribution Date	Contributor	Organization
2021-10-18	Parbati K. Manna provided detection methods and observed examples	Intel Corporation

Modifications

Previous Entry Names

Page Last Updated: July 16, 2024