CWE-327: Use of a Broken or Risky Cryptographic Algorithm

Weakness ID: 327
Vulnerability Mapping: ALLOWED (with careful review of mapping notes)
Abstraction: Class

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

Description

The product uses a broken or risky cryptographic algorithm or protocol.

Extended Description

Cryptographic algorithms are the methods by which data is scrambled to prevent observation or influence by unauthorized actors. Insecure cryptography can be exploited to expose sensitive information, modify data in unexpected ways, spoof identities of other users or devices, or other impacts.

It is very difficult to produce a secure algorithm, and even high-profile algorithms by accomplished cryptographic experts have been broken. Well-known techniques exist to break or weaken various kinds of cryptography. Accordingly, there are a small number of well-understood and heavily studied algorithms that should be used by most products. Using a non-standard or known-insecure algorithm is dangerous because a determined adversary may be able to break the algorithm and compromise whatever data has been protected.

Since the state of cryptography advances so rapidly, it is common for an algorithm to be considered "unsafe" even if it was once thought to be strong. This can happen when new attacks are discovered, or if computing power increases so much that the cryptographic algorithm no longer provides the amount of protection that was originally thought.

For a number of reasons, this weakness is even more challenging to manage with hardware deployment of cryptographic algorithms as opposed to software implementation. First, if a flaw is discovered with hardware-implemented cryptography, the flaw cannot be fixed in most cases without a recall of the product, because hardware is not easily replaceable like software. Second, because the hardware product is expected to work for years, the adversary's computing power will only increase over time.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	**Technical Impact:** *Read Application Data* The confidentiality of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.	
Integrity	**Technical Impact:** *Modify Application Data* The integrity of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.	
Accountability Non-Repudiation	**Technical Impact:** *Hide Activities* If the cryptographic algorithm is used to ensure the identity of the source of the data (such as digital signatures), then a broken algorithm will compromise this scheme and the source of the data cannot be proven.	

Potential Mitigations

Phase: Architecture and Design

Strategy: Libraries or Frameworks

When there is a need to store or transmit sensitive data, use strong, up-to-date cryptographic algorithms to encrypt that data. Select a well-vetted algorithm that is currently considered to be strong by experts in the field, and use well-tested implementations. As with all cryptographic mechanisms, the source code should be available for analysis.

For example, US government systems require FIPS 140-2 certification [REF-1192].

Do not develop custom or private cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. If the algorithm can be compromised if attackers find out how it works, then it is especially weak.

Periodically ensure that the cryptography has not become obsolete. 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]

Phase: Architecture and Design

Ensure that the design allows one cryptographic algorithm to be replaced with another in the next generation or version. Where possible, use wrappers to make the interfaces uniform. This will make it easier to upgrade to stronger algorithms. With hardware, design the product at the Intellectual Property (IP) level so that one cryptographic algorithm can be replaced with another in the next generation of the hardware product.

Effectiveness: Defense in Depth

Phase: Architecture and Design

Carefully manage and protect cryptographic keys (see CWE-320). If the keys can be guessed or stolen, then the strength of the cryptography itself is irrelevant.

Phase: Architecture and Design

Strategy: Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Industry-standard implementations will save development time and may be more likely to avoid errors that can occur during implementation of cryptographic algorithms. Consider the ESAPI Encryption feature.

Phases: Implementation; Architecture and Design

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 preventing common attacks.

Relationships

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

Nature	Type	ID	Name
ChildOf	P	693	Protection Mechanism Failure
ParentOf	B	328	Use of Weak Hash
ParentOf	V	780	Use of RSA Algorithm without OAEP
ParentOf	B	1240	Use of a Cryptographic Primitive with a Risky Implementation
PeerOf	C	311	Missing Encryption of Sensitive Data
PeerOf	B	301	Reflection Attack in an Authentication Protocol
CanFollow	B	208	Observable Timing Discrepancy

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Relevant to the view "Architectural Concepts" (CWE-1008)

Modes Of Introduction

Phase	Note
Architecture and Design	COMMISSION: This weakness refers to an incorrect design related to an architectural security tactic.
Implementation	With hardware, the Architecture or Design Phase might start with compliant cryptography, but it is replaced with a non-compliant crypto during the later Implementation phase due to implementation constraints (e.g., not enough entropy to make it function properly, or not enough silicon real estate available to implement). Or, in rare cases (especially for long projects that span over years), the Architecture specifications might start with cryptography that was originally compliant at the time the Architectural specs were written, but over the time it became non-compliant due to progress made in attacking the crypto.

Applicable Platforms

Languages

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

Verilog *(Undetermined Prevalence)*

VHDL *(Undetermined Prevalence)*

Technologies

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

Class: ICS/OT *(Undetermined Prevalence)*

Likelihood Of Exploit

High

Demonstrative Examples

Example 1

These code examples use the Data Encryption Standard (DES).

Example Language: **C** *(bad code)*

```c
EVP_des_ecb();
```

Example Language: **Java** *(bad code)*

```java
Cipher des=Cipher.getInstance("DES...");
des.initEncrypt(key2);
```

Example Language: **PHP** *(bad code)*

```php
function encryptPassword($password){
  $iv_size = mcrypt_get_iv_size(MCRYPT_DES, MCRYPT_MODE_ECB);
  $iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
  $key = "This is a password encryption key";
  $encryptedPassword = mcrypt_encrypt(MCRYPT_DES, $key, $password, MCRYPT_MODE_ECB, $iv);
  return $encryptedPassword;
}
```

Once considered a strong algorithm, DES now regarded as insufficient for many applications. It has been replaced by Advanced Encryption Standard (AES).

Example 2

Suppose a chip manufacturer decides to implement a hashing scheme for verifying integrity property of certain bitstream, and it chooses to implement a SHA1 hardware accelerator for to implement the scheme.

Example Language: **Other** *(bad code)*

```
The manufacturer chooses a SHA1 hardware accelerator for to implement the scheme because it already has a working SHA1 Intellectual Property (IP) that the manufacturer had created and used earlier, so this reuse of IP saves design cost.
```

However, SHA1 was theoretically broken in 2005 and practically broken in 2017 at a cost of $110K. This means an attacker with access to cloud-rented computing power will now be able to provide a malicious bitstream with the same hash value, thereby defeating the purpose for which the hash was used.

This issue could have been avoided with better design.

Example Language: **Other** *(good code)*

```
The manufacturer could have chosen a cryptographic solution that is recommended by the wide security community (including standard-setting bodies like NIST) and is not expected to be broken (or even better, weakened) within the reasonable life expectancy of the hardware product. In this case, the architects could have used SHA-2 or SHA-3, even if it meant that such choice would cost extra.
```

Example 3

In 2022, the OT:ICEFALL study examined products by 10 different Operational Technology (OT) vendors. The researchers reported 56 vulnerabilities and said that the products were "insecure by design" [REF-1283]. If exploited, these vulnerabilities often allowed adversaries to change how the products operated, ranging from denial of service to changing the code that the products executed. Since these products were often used in industries such as power, electrical, water, and others, there could even be safety implications.

Multiple OT products used weak cryptography.

Observed Examples

Reference	Description
CVE-2022-30273	SCADA-based protocol supports a legacy encryption mode that uses Tiny Encryption Algorithm (TEA) in ECB mode, which leaks patterns in messages and cannot protect integrity
CVE-2022-30320	Programmable Logic Controller (PLC) uses a protocol with a cryptographically insecure hashing algorithm for passwords.
CVE-2008-3775	Product uses "ROT-25" to obfuscate the password in the registry.
CVE-2007-4150	product only uses "XOR" to obfuscate sensitive data
CVE-2007-5460	product only uses "XOR" and a fixed key to obfuscate sensitive data
CVE-2005-4860	Product substitutes characters with other characters in a fixed way, and also leaves certain input characters unchanged.
CVE-2002-2058	Attackers can infer private IP addresses by dividing each octet by the MD5 hash of '20'.
CVE-2008-3188	Product uses DES when MD5 has been specified in the configuration, resulting in weaker-than-expected password hashes.
CVE-2005-2946	Default configuration of product uses MD5 instead of stronger algorithms that are available, simplifying forgery of certificates.
CVE-2007-6013	Product uses the hash of a hash for authentication, allowing attackers to gain privileges if they can obtain the original hash.

Detection Methods

Automated Analysis

Automated methods may be useful for recognizing commonly-used libraries or features that have become obsolete.

Effectiveness: Moderate

Note: False negatives may occur if the tool is not aware of the cryptographic libraries in use, or if custom cryptography is being used.

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Note: These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:

- Bytecode Weakness Analysis - including disassembler + source code weakness analysis
- Binary Weakness Analysis - including disassembler + source code weakness analysis
- Binary / Bytecode simple extractor - strings, ELF readers, etc.

Effectiveness: SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:

- Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness: SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:

- Web Application Scanner
- Web Services Scanner
- Database Scanners

Effectiveness: SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful:

Highly cost effective:

- Man-in-the-middle attack tool

Cost effective for partial coverage:

- Framework-based Fuzzer
- Automated Monitored Execution
- Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness: High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful:

Highly cost effective:

- Manual Source Code Review (not inspections)

Cost effective for partial coverage:

- Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness: High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful:

Highly cost effective:

- Source code Weakness Analyzer
- Context-configured Source Code Weakness Analyzer

Effectiveness: High

Automated Static Analysis

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:

- Configuration Checker

Effectiveness: SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful:

Highly cost effective:

- Formal Methods / Correct-By-Construction

Cost effective for partial coverage:

- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness: High

Memberships

Nature	Type	ID	Name
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage
MemberOf	C	753	2009 Top 25 - Porous Defenses
MemberOf	C	803	2010 Top 25 - Porous Defenses
MemberOf	C	816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage
MemberOf	C	866	2011 Top 25 - Porous Defenses
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)
MemberOf	V	884	CWE Cross-section
MemberOf	C	934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure
MemberOf	C	958	SFP Secondary Cluster: Broken Cryptography
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure
MemberOf	C	1131	CISQ Quality Measures (2016) - Security
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures
MemberOf	C	1366	ICS Communications: Frail Security in Protocols
MemberOf	C	1402	Comprehensive Categorization: Encryption

Vulnerability Mapping Notes

Usage: ALLOWED-WITH-REVIEW *(this CWE ID could be used to map to real-world vulnerabilities in limited situations requiring careful review)*

Reason: Abstraction

Rationale:
This CWE entry is a Class and might have Base-level children that would be more appropriate

Comments:
Examine children of this entry to see if there is a better fit

Notes

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.

Maintenance

The Taxonomy_Mappings to ISA/IEC 62443 were added in CWE 4.10, but they are still under review and might change in future CWE versions. These draft mappings were performed by members of the "Mapping CWE to 62443" subgroup of the CWE-CAPEC ICS/OT Special Interest Group (SIG), and their work is incomplete as of CWE 4.10. The mappings are included to facilitate discussion and review by the broader ICS/OT community, and they are likely to change in future CWE versions.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using a broken or risky cryptographic algorithm
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage
CERT C Secure Coding	MSC30-C	CWE More Abstract	Do not use the rand() function for generating pseudorandom numbers
CERT C Secure Coding	MSC32-C	CWE More Abstract	Properly seed pseudorandom number generators
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers
OMG ASCSM	ASCSM-CWE-327		
ISA/IEC 62443	Part 3-3		Req SR 4.3
ISA/IEC 62443	Part 4-2		Req CR 4.3

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-20	Encryption Brute Forcing
CAPEC-459	Creating a Rogue Certification Authority Certificate
CAPEC-473	Signature Spoof
CAPEC-475	Signature Spoofing by Improper Validation
CAPEC-608	Cryptanalysis of Cellular Encryption
CAPEC-614	Rooting SIM Cards
CAPEC-97	Cryptanalysis

References

[REF-280] Bruce Schneier. "Applied Cryptography". John Wiley & Sons. 1996. <https://www.schneier.com/books/applied-cryptography>. *URL validated: 2023-04-07*.

[REF-281] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. "Handbook of Applied Cryptography". 1996-10. <https://cacr.uwaterloo.ca/hac/>. *URL validated: 2023-04-07*.

[REF-282] C Matthew Curtin. "Avoiding bogus encryption products: Snake Oil FAQ". 1998-04-10. <http://www.faqs.org/faqs/cryptography-faq/snake-oil/>.

[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-284] Paul F. Roberts. "Microsoft Scraps Old Encryption in New Code". 2005-09-15. <https://www.eweek.com/security/microsoft-scraps-old-encryption-in-new-code/>. *URL validated: 2023-04-07*.

[REF-7] Michael Howard and David LeBlanc. "Writing Secure Code". Chapter 8, "Cryptographic Foibles" Page 259. 2nd Edition. Microsoft Press. 2002-12-04. <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223>.

[REF-44] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 21: Using the Wrong Cryptography." Page 315. McGraw-Hill. 2010.

[REF-287] Johannes Ullrich. "Top 25 Series - Rank 24 - Use of a Broken or Risky Cryptographic Algorithm". SANS Software Security Institute. 2010-03-25. <https://www.sans.org/blog/top-25-series-use-of-a-broken-or-risky-cryptographic-algorithm/>. *URL validated: 2023-04-07*.

[REF-62] Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". Chapter 2, "Insufficient or Obsolete Encryption", Page 44. 1st Edition. Addison Wesley. 2006.

[REF-962] Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". ASCSM-CWE-327. 2016-01. <http://www.omg.org/spec/ASCSM/1.0/>.

[REF-18] Secure Software, Inc.. "The CLASP Application Security Process". 2005. <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf>.

[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-1283] Forescout Vedere Labs. "OT:ICEFALL: The legacy of "insecure by design" and its implications for certifications and risk management". 2022-06-20. <https://www.forescout.com/resources/ot-icefall-report/>.

Content History

Submissions

Submission Date	Submitter	Organization
2006-07-19 *(CWE Draft 3, 2006-07-19)*	CLASP	

Contributions

Contribution Date	Contributor	Organization
2019-12-10	Parbati K. Manna Provide a hardware-specific submission whose contents were integrated into this entry, affecting extended description, applicable platforms, demonstrative examples, and mitigations	Intel Corporation

Modifications

Previous Entry Names

Page Last Updated: July 16, 2024