Safety and Authentication

Martin Lottermoser
http://home.htp-tel.de/lottermose2

Version 1.5 (2015-09-07)

Contents

1 Introduction

In recent years, IT security incidents have increased in number or have become more publically visible. This has resulted in pressure on organizations selling IT components or offering IT-based services to control such threats. While many developed countries have had appropriate conventions or regulations in place for at least two decades, this typically applied only to governmental IT infrastructure. But now some countries (e.g., Germany [1]) have started to encourage or enforce certain IT security measures also in commercially available products, in particular those where a disruption of "normal" operation affects a large number of people or where a failure would have unacceptable consequences (critical infrastructure).

This does not seem much of a problem for organizations which offer "safe" (no harm to persons) components or services[1] as they have already had to take measures in their systems and procedures which provide a substantial level of security, mainly because the primary means for achieving safety (isolation of parts and comparing results between redundant parts) makes malicious attacks much more difficult. However, this aspect is partially counteracted by an increasing demand for remotely controlled systems, due to the desire to reduce system reaction times and to eliminate operator errors by automation[2]; the larger the network connecting such components becomes, the more difficult it is to control access to the network. This means

[1] My personal interest is mostly directed at railway signalling systems and that in the context of European standards [3, 4, 5, 6], but I believe that the main body of my argument is independent of that.
[2] Automation is also attractive because a reduction in personnel may lower operating costs.

that at least for data communication, isolating the parts is no longer a feasible means for achieving safety and hence such systems become exposed to security threats with possibly safety-related consequences.

Operators and vendors of safety-related systems therefore have had to consider how to counter such threats. I've seen discussions on this topic to focus on two aspects of solutions:

1. Full use of cryptography (confidentiality, authentication, and integrity) for all data exchanged
2. Cryptographic endpoints are added to existing parts by placing the former in non-safe components outside of safety-related (SIL > 0) areas but still within an access-controlled environment, similar to existing solutions for connecting access-controlled networks over larger distances through VPN-based technology ("access protection layer" [5, figure C.3]).

I consider both theses aspects of solutions to be faulty because, in my opinion, authentication is safety-related, confidentiality is not, and the usual cryptographic measures for ensuring authenticity already provide integrity.

The main body of this article gives the reasons for my opinion and points out some consequences for implementations.

2 Why We Shouldn't Focus on Confidentiality

Messages exchanged within a safety-related system are usually concerned with the current state of the system and commands to change it; if an attacker gains knowledge of these data, this by itself has no consequence at all for the safety of the system.

But these data are also usually very short and simple and do not contain confidential or vendor-proprietary information, hence there is no need for the system's vendor to protect it. Hiding this information from listeners might look advantageous to the operator, but I consider that to be similar to the common "security by obscurity" fallacy, meaning that it doesn't work:

- One aspect of the increasing number of remotely-controlled systems is that components from different vendors have to interact. This means that the interfaces and hence the structure of the data exchanged are known to more than one organization. The operator can't expect that an attacker will not get access to these specifications at the source.
- Protecting information about the current state of the system cannot be completely achieved by encrypting the corresponding messages. (If you don't want an attacker to know whether a particular railway signal will let a train pass, how do you prevent him/her from just looking at it?)

In addition, encrypting the data traffic has the disadvantage that monitoring systems must be cryptographically aware and have to be administered accordingly; this adds to operating costs and prevents the most confidential solution where the key needed for decryption is only known to the recipient. Note also that safety-related components are usually embedded systems with limited computing resources; encrypting messages might occupy a substantial part of these resources.

This should, however, not be understood as a claim that confidentiality should be omitted from safety-related systems. I'm just saying that (a) it is not related to safety, (b) its security benefit is low, and (c) its implementation cost is high. If we can get confidentiality with small effort, we might as well take it, but other aspects of a safety-related system are much more important.

3 Why Authentication Provides Integrity

If an entity receives a message[3] claiming to come from a particular sender and the sender is unknown or the medium used for transmission might have been tampered with, the recipient should determine whether the claim is true (authentication).

The only cryptographically trustworthy and at the same time feasible methods for authentication I know of are based on a suitable public-private key pair [2]. The sender uses his/her private key to encode some

[3] I'm only considering connection-less communication in this article. Connection-oriented methods of communication are subject to the same problems, but restricted to the phase of establishing a connection; they then typically try to switch to simpler procedures for a certain interval of time, repeating this process until the connection is terminated.

message-related data and transmits the result to the recipient; the recipient applies the sender's public key to the data and verifies the decoded data against the message. If the check is successful, the recipient knows that only someone with knowledge of that particular private key can have created the message. The final step is then to ensure that this key is indeed associated with the claiming entity; this can be achieved through a trust database of key-identity associations for which the recipient is certain that knowledge of the private key has not spread beyond the owning entity.[4]

In principle it seems possible for the sender to encode the entire message and to only transmit those data. However, the two transformations need non-negligible computing resources scaling with the message's size, and the recipient has no other data against which the decoded message can be checked. Such a procedure can still be feasible if both partners have sufficient resources and the message has enough structure (e.g., a "magic" string or a CRC field) to permit the recipient to perform a validity check of sufficiently high discernment on the decoded message alone.

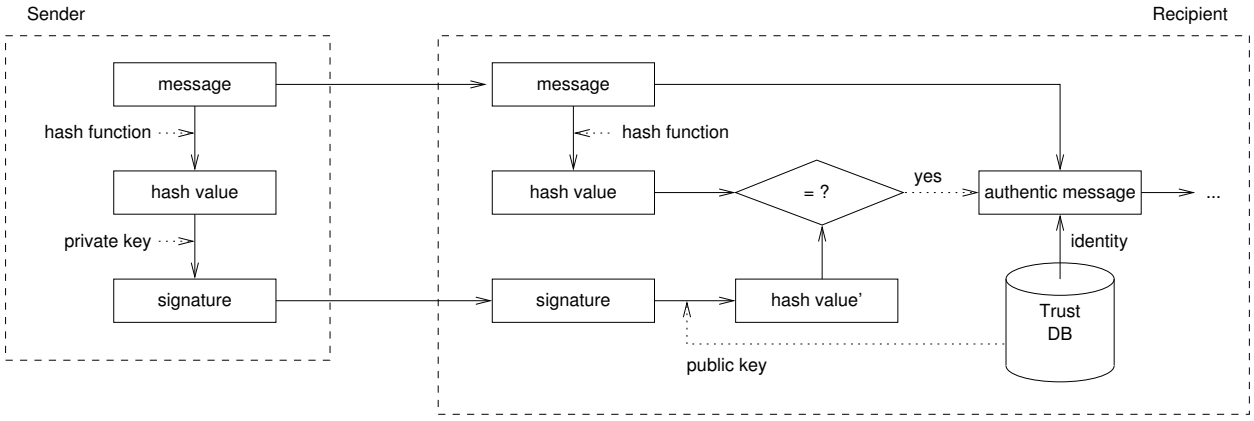

Figure 1: Verification of cryptographic signatures

However, it is *much* more efficient to send the message in unencoded form, accompanied by an encoded hash value derived from the message (cryptographic signature, also known as "message authentication code (MAC)"; see figure 1). The recipient determines the hash value the received message has and compares it with the decoded value of the signature. If they are equal and the hash function is cryptographically strong (i.e., it is considered practically impossible that an attacker can have substitued a different message with the same hash value), the recipient can conclude that the owner of the private key has indeed applied the hash function to *this* message.

But this means that a successful process of authentication already has checked that the message has not been modified in transit! The hash value underlying the signature is an integrity checksum and could be used for that purpose even if the key-identity association were unknown. Hence such a check for authenticity already performs a check for integrity.

For simplicity I shall also assume that the signature is sufficiently good at providing an integrity check at the level of safety needed ("cryptographic safety code" [5, appendix C.2]). This is a requirement on the hash function, but as cryptographic hash functions have to be sufficiently robust against deliberate attacks I would expect that to hold at even higher level against random transmission errors. Should this assumption turn out not to be true, one can either choose another hash function or include a safety code within the unencoded data.

4 Why is Authentication Safety-Related?

Consider a system which controls the flow of a fluid into and out of a vat. There are flow sensors and valves at both ends and it is the job of the system to ensure that the vat does not overflow. If the system's controlling

[4] Although it is possible to fill this database once in a secure environment, the usual method is to only provide a small number of trusted associations initially and to extend the list later by means of "certificates", i.e., messages containing key-identity associations of previously untrusted entities but signed by an already trusted one. Such key management procedures are outside the scope of this article.

entity now incorrectly identifies the sensor's message from the entry to come from the sensor at the exit and *vice versa*, the system will instead ensure that the amount of fluid in the vat does not drop below a certain level, which means that the vat might now overflow. If the fluid has dangerous properties, preventing the overflow will have to be a safety requirement. A misidentification of the source of a message is then a cause for a safety hazard, hence the process of authentication has to be a safety function.

Note that this argument does not even consider security threats: a failure in authentication may already lead to a safety hazard in systems where access to the system is strictly controlled.

5 Consequences for Performing Authentication

If authentication is a safety function, it must be performed in a part of the system which is safety-related, i.e., in a part with SIL > 0.

In the presence of security threats, authentication must be cryptographically protected as described in section 3. As a failure at this step still may lead to a safety hazard, at least this part of cryptographic protection has to satisfy safety requirements and therefore also has to be executed in safety-related equipment.

Note that the reference architecture of EN 50159 [5, figure 1] assumes that "safety related cryptographic techniques" may be performed *outside* safety-related equipment, in particular outside the safety-related transmission function (which typically performs basic authentication based on source identifiers). In view of security threats, I consider that split to be insufficiently safe.[5]

6 Possible Solutions

6.1 Execution Model for Safety-Related Systems

In what follows, I'm going to assume an execution model with two components providing a "safe core" (see figure 2):

1. An environment (hardware and firmware) which provides a platform for the execution of software

   Given the desired SIL s for the system, this platform must have the property that, executing any software of SSIL s, the resulting system will achieve SIL s.
   Basically, the execution environment must be capable of guaranteeing that correct software would execute correctly, at the level of safety desired. This typically means that such a platform will contain
   (a) one or more processors (CPUs) and
   (b) certain amounts of volatile and
   (c) non-volatile memory,

   together with measures to guard against hardware failures or systematic errors in the parts making up this platform.
   The execution environment must also be secure in the sense that it can't be tampered with. This typically entails access control measures like locked cabinets with intrusion detection for the hardware and steps to protect against modifications of the software to be executed.
2. Software of SSIL s executing on the platform

The key point here is that the software component can't guard itself completely against failures in those hardware components (CPU and memory) which are absolutely essential for executing software, because, if they fail, any checks performed on them by the software may give incorrect results. Therefore these essential parts must be sufficiently safe and secure by themselves. However, I should like to point out two restrictions on this requirement.

The first is that we need safety and security only as a service offered at the interface to the software, i.e., it is not necessary for all parts of the execution environment to be safe or secure. Thus these requirements restrict the execution environment as a platform, and not every part of its implementation.

[5] Note also that, in view of the increasing sophistication of security threats, the assumption that a local-area network can be access-controlled becomes decreasingly convincing. For example, solutions like an "access protection layer" [5, figure C.3] are insufficient to counter attacks which might take over one of the entry points into the network (single point of failure).

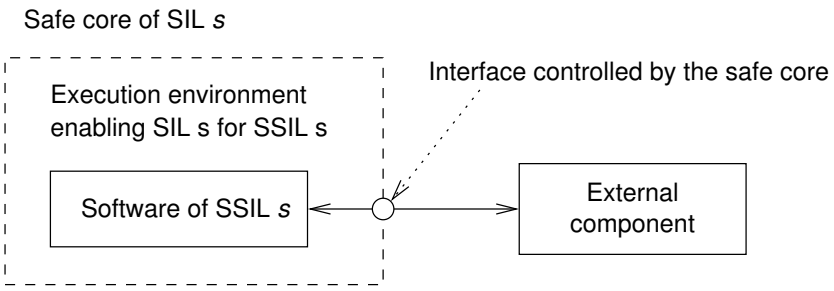

Figure 2: Structure and environment of a safe core

The second point is that these properties are only necessary for the execution platform and not for all the hardware components accessed by the software. Safety-related software may directly communicate with another component which is not part of the execution platform and which might even not be safe, provided the platform and the software can sufficiently control the influence of these external components on the safe core. A typical example is a passive component which accepts commands from the core and in return offers data to be fetched whenever the core is ready. The ability to control the interfaces in this or a similar manner is an essential and safety-related requirement for all the core's external interfaces.

6.2 Everything in the Safe Core

If authentication is a safety function, the simplest solution for satisfying the safety requirements is obviously to perform the entire process of authentication (see figure 1) in the safe core (see figure 2) where we have all the components needed at our disposal, including non-volatile memory for the trust database.

However, two aspects complicate matters:

- Safe cores are often implemented in embedded systems, processors for embedded systems are usually *much* slower than processors for PCs, and cryptographic operations (in particular with public or private keys) are time-consuming.
- Authentication is only the recipient's part in successfully passing a message known to be authentic: we also have to find a way for storing the sender's private key in such a manner that it does not incur a security leak and hence does not pose a safety risk.

Both of these aspects suggest that a separate component dedicated to cryptographic operations and hiding a secret key might be advantageous; such cryptographic processors are commercially available.

6.3 Using a Safe Cryptographic Processor

If we had a safe cryptographic processor, we could perform the entire process of signature verification (figure 1) within that component. The processor could be given a message and its signature and could either return an authentication failure or indicate that the message is authentic. Again, however, reality intervenes:

- If the cryptographic processor is a separate external component like shown in figure 2, we have at least to guard against transmission errors[6] on the line between the safe core and the cryptographic processor. (If the line were safe, this connection as well as the cryptographic processor could be considered to be part of the safe core, and that is not much different from the solution already considered in section 6.2.)
- Commercially available cryptographic processors are unlikely to be certified as safe in the first place.

6.4 Using a Non-Safe Cryptographic Processor

The real challenge is therefore how to use a non-safe (but hopefully secure) cryptographic processor such that authentication is still safe.

[6] If access to the line cannot be strictly controlled, we also would have to guard against man-in-the-middle attacks. Hence we would have to perform authentication on this line as well, leading to a chicken-egg problem.

The key idea for the solution I wish to propose is that no decision which is safety-related and message-specific may be delegated to the cryptographic processor. Therefore, the comparison between the hash value of the message and the decoded hash value from the signature (see figure 1) must be performed by the safe core. We can (and should) delegate the decoding of the signature to the processor, but computing the hash value from the message is again something we can't safely trust to the processor[7] because, due to some failure, the second of these two operations might just lead to the processor returning the result of the first, thus leading to a message which is always declared to be authentic although it might not be.

I'm therefore assuming a cryptographic processor with the following interface:

1. The processor can be directly accessed from the safe execution environment and only from it. Thus the processor is an access-controlled part of the safe core, but it need not be safety-related itself.
2. The processor offers a command to create an unpredictable new secret key which will be stored securely within the processor.
3. The processor can be queried for the public key belonging to its current secret key.
4. The processor offers commands for encoding bit strings with its secret key, at least for the lengths intended for the hash function.
5. The processor offers commands for decoding bit strings with public keys also given as parameters to the commands; again, this must be possible at least for the lengths intended for the hash function and for the type of keys used by other participants.

Given a processor with these properties,

- initial trust databases can be built in a secure environment,
- a sender can have the processor encode the hash value without divulging the secret key, and
- the recipient can speed up decoding the signature by offloading the task to the processor.

Computing the hash value on both sides, sender and recipient, should be executed in the safe core, and the final task of comparing the two hash values must be executed there. Note also that the trust database must be held in the safe core.

With such an architecture, failures in the cryptographic processor or in communicating with it can affect only the transformations of the signature:

- If the processor at the sender fails, the message will have an invalid signature, will be (correctly) classified as invalid by the recipient, and will be discarded.
- If a recipient receives a valid message and the processor at the recipient fails, the recipient will incorrectly consider the message to be invalid and will discard it. This is effectively the same as the message not reaching the recipient at all and is a situation the recipient must be able to handle anyway.
- If a recipient receives an invalid message and the processor at the recipient fails, the decoded hash value returned by the processor might erroneously agree with the value derived from the message. This is a risk, but given the length and robustness of typical cryptographic hash functions and the fact that the processor has not seen the unencoded message this event is wildly unlikely.

Thus it is indeed possible to perform authentication safely with a non-safe cryptographic processor.

7 Conclusion

In summary, I consider the following statements to be the salient points of this article:

1. Confidentiality is not safety-related.
2. Correct authentication is a safety requirement and the function providing it must therefore be executed with SIL > 0.
3. It is possible to use a non-safe cryptographic processor to implement a safe authentication function, but the processor must be placed within the access-controlled region of the safe core and it must not gain direct knowledge of the content of messages.

[7] But if we had two safely independent cryptographic processors, we might use one for decoding the signature and the other for computing the hash value. Whether that is useful will largely depend on the computing power available and is therefore not pursued further in this article.

Appendix

References

[1] Bundesrepublik Deutschland. Gesetz zur Erhöhung der Sicherheit informationstechnischer Systeme (IT-Sicherheitsgesetz) — Vom 17. Juli 2015. *Bundesgesetzblatt*, Jahrgang 2015, Teil I(31):1324–1331, July 2015.

[2] Whitfield Diffie, Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.
URL `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1055638`

[3] European Committee for Electrotechnical Standardization (CENELEC), Brussels. *Railway applications — The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS)*, September 1999. EN 50126, English version. In May 2006, this standard was renamed to EN 50126-1.

[4] European Committee for Electrotechnical Standardization (CENELEC), Brussels. *Railway applications — Communication, signalling and processing systems — Safety related electronic systems for signalling*, February 2003. EN 50129, English version.

[5] European Committee for Electrotechnical Standardization (CENELEC), Brussels (Belgium). *Railway applications — Communication, signalling, and processing systems — Safety-related communication in transmission systems*, September 2010. EN 50159, English version.

[6] European Committee for Electrotechnical Standardization (CENELEC), Brussels (Belgium). *Railway applications — Communication, signalling and processing systems — Software for railway control and protection systems*, June 2011. European Standard EN 50128, English version.

Abbreviations

CPU	Central processing unit
CRC	Cyclic redundancy check
DB	Database
IT	Information technology
PC	Personal computer
SIL	Safety integrity level
SSIL	Software SIL
VPN	Virtual private network

History of Changes

1.4	2015-08-27	First distributed version
1.5	2015-09-07	Added access-control requirements to the concept of an execution platform; minor editorial changes for clarification.

Copyright and License

Address:

Martin Lottermoser
Greifswaldstr. 28
38124 Braunschweig
Germany

Although I encountered the problems described at the outset of this article in the course of work for my employer, solving them was and is not part of my responsibilities: I'm not employed as a systems architect or software architect, nor am I involved in any aspect of safety assessment or safety management. I've therefore created this article entirely in my spare time and at home (actually while I was on holiday), and my employers (past and present) are not responsible for any statements I've made here.