# Security Protocol Deployment Risk

Simon N. Foley[1], Giampaolo Bella[2,3], and Stefano Bistarelli[4,5]

[1] Department of Computer Science, University College Cork, Ireland
[2] SAP Research, Mougins, France
[3] Dipartimento di Matematica e Informatica, Università di Catania, Italy
[4] Dipartimento di Scienze, Universitá degli Studi "G. D'Annunzio", Pescara, Italy
[5] Istituto di Informatica e Telematica, CNR, Pisa, Italy

Security protocol participants are software and/or hardware agents that are—as with any system—potentially vulnerable to failure. Protocol analysis should extend not just to an analysis of the protocol specification, but also to its implementation and configuration in its target environment. However, an in-depth formal analysis that considers the behavior and interaction of all components in their environment is not feasible in practice.

This paper considers the analysis of protocol deployment rather than implementation. Instead of concentrating on detailed semantics and formal verification of the protocol and implementation, we are concerned more with with the ability to trace, at a practical level of abstraction, how the protocol deployment, that is, the configuration of the protocol components, relate to each other and the overall protocol goals. We believe that a complete security verification of a system is not currently achievable in practice and seek some degree of useful feedback from an analysis that a particular protocol deployment is reasonable.

## 1 Vulnerabilities in Deployed Protocols

The deployment of a protocol is a collection of interacting software/hardware components that collectively implement the protocol in its environment. Components range from agents that fully implement a protocol principal, for example, an authentication server running on a bastion host, to entities that partially support the principal, for example, a hardware token authenticator. These components may suffer a variety of vulnerabilities.

*Design Vulnerabilities* are errors in the underlying design of the protocol whereby it fails to achieve its intended security goals. For example, a replay attack on the protocol that enables a principal to masquerade as a different principal [2, 5].

*Implementation Vulnerabilities* The software that implements the protocol may be vulnerable to attack. For example, a component with a buffer-overflow vulnerability may be vulnerable to a stack-smashing attack from another principal that could lead to, for example, disclosure of keys, replay attack, etc. Improper implementation of a client may also facilitate an attack, for example, a Kerberos client that does not discard the user-password (long-term key) once the ticket-granting ticket/key is obtained from the authentication server.

*Configuration Vulnerabilities* are a consequence of any improper configuration of the component that might lead to its compromise. For example, an authentication server that stores long-term cleartext keys in a file that is accessible by all users. An authentication client that incorporates a hardware token authenticator might be considered less vulnerable than a client that relied on a fixed password for its long-term secret.

*System Vulnerabilities* The platform that hosts a component may itself be vulnerable to attack. For example, deploying an authentication server on a system alongside a web-server would be considered a poor choice. Similarly, hosting an authentication server on a hardened bastion host managed by a competent administrator might be considered less vulnerable to attack than hosting it on an out-of-the-box workstation that has outstanding software patches.

There are many further vulnerabilities that could be considered, for example, cryptographic API vulnerabilities, vulnerabilities in weak cryptographic operations, and so forth. We do not intend to analyze or model vulnerabilities per-se, but instead reflect vulnerability and threat in terms of an abstract degree of *confidence* that we have in a component's secure and proper operation. The degree of confidence of a component can be based on evidence (a 'correct' protocol running on a bastion host) and/or subjective views (the administrator of this system is considered to be incompetent). We do not intend to only consider the deployment of components for which we have complete competence: we are interested in determining whether protocols that are deployed as combinations of good, bad and indifferent components are good enough.

## 2 Protocol Deployment Risk

Degree of confidence [6] is defined in terms of the set $\mathcal{C}$ of confidence levels that is ordered under $\leq$. This is interpreted as: given $a, b \in \mathcal{C}$ then $a \leq b$ means that we have no less confidence in a component with confidence level $b$ than a component with confidence level $a$. For example, we might define confidence levels $\mathsf{lo} < \mathsf{med} < \mathsf{hi}$, with the obvious interpretation, for example, we have $\mathsf{lo}$ confidence in a component executing on an open-access workstation and have $\mathsf{hi}$ confidence in an authentication server executing on a bastion host. In this paper a c-semiring [3] is used to represent degree of confidence. This provides a variety of measures for confidence, ranging from simple orderings such as $\mathsf{lo} < \mathsf{med} < \mathsf{hi}$ to numeric measures such as fuzzy and numeric weightings.

We consider confidence in the context of the cryptographic keys that the protocol components are expected to manage. For example, we might have a high degree of confidence that an authentication server component, with few known vulnerabilities, properly secures the keys that it manages; this might reflect a confidence that it will not leak keys to the public channel.

Another ordering $\leq$ is defined over the class of cryptographic keys referenced in the protocol to reflect the degree to which they require protection [1]. For example, a long-term password would be considered more critical than a short-term

session key, since compromise of the latter is a once-off threat, while compromise of the former may lead to repeated attacks. The set $\mathcal{K}$ of key classifications forms a lattice under $\leq$ with a lower bound element public that represents the public channel.

**Example 1** An authentication and key distribution server Trent manages long term and session keys. Trent shares long term keys with principals Alice and Bob, and issues short-term session keys. The following Needham-Schroder style protocol is followed to acheive key exchange. For the moment we do not consider the components that Alice, Bob or Trent might comprise.

$$\begin{aligned}
&\text{Msg } 1 : A \rightarrow T : A, B, N_A \\
&\text{Msg } 2 : T \rightarrow A : \{\!|K_{AB}, A, B, N_A, T_T|\!\}_{K_A}, \{\!|K_{AB}, A, B, T_T|\!\}_{K_B} \\
&\text{Msg } 3 : A \rightarrow B : \{\!|K_{AB}, A, B, T_T|\!\}_{K_B}
\end{aligned}$$

Three symmetric keys $K_A$, $K_B$ and $K_{AB}$ are used in this protocol, and we define corresponding key classes levels Ka, Kb and Kab; class public corresponds to the public channel.

Figure 1 defines the class ordering $(\mathcal{K}, \leq)$. Long-term keys are used to transfer session keys and, therefore, we have Kab $<$ Ka and Kab $<$ Kb where Ka and Kb are disjoint (cannot deduce one from the other), and $\top$ defines the universal upper bound on the ordering. This reflects an assumption in the protocol that long-term keys are considered more security-critical than session keys: in the absence of ephemeral keys, loss of the long-term key implies loss of the short-term keys (but not vice-versa). $\triangle$

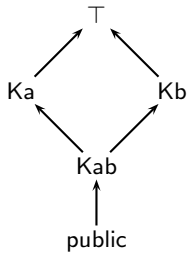


**Fig. 1.** Key Classification Ordering $(\mathcal{K}, \leq)$

The key classification ordering defines the flow constraints between keys, for example, long term key Ka information should not be permitted to flow to session key Kab information. Rather than relying on a binary interpretation of how key related information may/may not flow, we take a qualitative-based approach that is similar to the notion of assurance in [9, 7]. Define

$$minConf : \mathcal{K} \times \mathcal{K} \rightarrow \mathcal{C}$$

where $minConf(k, k')$ defines the minimum confidence required across the protocol deployment that a key of class $k$ does not flow to class $k'$. The lower bound $\bot$ in the confidence c-semiring is interpreted as no flow restriction and thus, if $k \leq k'$, then $minConf(k, k') = \bot$ in the confidence c-semiring, since, in this case, the flow is permitted.

**Example 2** Consider the set $(\mathcal{C}, \leq)$ of confidence ratings $\mathsf{nil} < \mathsf{lo} < \mathsf{med} < \mathsf{hi}$, where $\mathsf{nil}$ is the lower bound and corresponds to no flow restriction. Given the protocol in Example 1 then there are no flow restrictions on $\mathsf{public}$ as a source, that is, $minConf(\mathsf{public}, x) = \mathsf{nil}$ for any $x \in \mathcal{K}$. By reflexivity of $(\mathcal{K}, \leq)$ we also have $minConf(x, x) = \mathsf{nil}$ for $x \in \mathcal{K}$.

Key class $\top$ represents the most security-critical key (aggregate of the long-term keys) and we define:

$$\begin{array}{l|l} minConf([\top, \mathsf{public}]) = \mathsf{hi} & minConf([\top, \mathsf{Kab}]) = \mathsf{hi} \\ minConf([\top, \mathsf{Ka}]) = \mathsf{med} & minConf([\top, \mathsf{Kb}]) = \mathsf{med} \end{array}$$

An authorization server that manages both long-term keys and the session key (interval $[\mathsf{Kab}, \top]$ requires greater confidence ($\mathsf{hi}$) in its protection than its clients (confidence $\mathsf{med}$) that manage one long-term key and session key; in the former, we require greater confidence that the keys cannot be leaked. We have:

$$\begin{array}{l|l} minConf([\mathsf{Ka}, \mathsf{Kab}]) = \mathsf{med} & minConf([\mathsf{Ka}, \mathsf{Kb}]) = \mathsf{hi} \\ minConf([\mathsf{Ka}, \mathsf{public}]) = \mathsf{med} & minConf([\mathsf{Ka}, \top]) = \mathsf{nil} \end{array}$$

with a similar definition for $\mathsf{Kb}$. Protecting only a session key requires less confidence:

$$\begin{array}{l|l} minConf([\mathsf{Kab}, \mathsf{public}]) = \mathsf{lo} & minConf([\mathsf{Kab}, \mathsf{Ka}]) = \mathsf{nil} \\ minConf([\mathsf{Kab}, \mathsf{Kb}]) = \mathsf{nil} & minConf([\mathsf{Kab}, \top]) = \mathsf{nil} \end{array}$$

△

Each protocol component can be regarded as managing a number of different kinds of keys. The authentication server in Example 1 manages $\mathsf{Ka}$, $\mathsf{Kb}$ and $\mathsf{Kab}$ key and the client manages $\mathsf{Kab}$ and $\mathsf{Ka}$ keys. Every component $c$, is bound to an interval of the key classification lattice, where $int(c) = [l, h] \in \mathcal{K} \times \mathcal{K}$, and $l \leq h$ is interpreted as follows:

- $l$ is the lowest classification key that the component encrypts messages with. Here encrypt is like 'send' in terms of information flow between key classifications.
- $h$ is the highest classification key that the component decrypts messages with. Here decrypt is like 'receive' in terms of information flow between key classifications.

We also write $int(c) = [int_\bot(c), int_\top(c)]$.

Each protocol component $c$ also has a confidence rating given as $rating(c)$ that also reflects the minimum effort that would be required by an attacker to

compromise component $c$. For example, we might have high confidence in the authentication service in Example 1 that is deployed on a hardened selinux server, but have low confidence in the client component implemented as freeware and running on an open-access workstation. In the case of the authentication server we are confident that keys will not be leaked nor messages encrypted/decrypted in a way that does not follow the protocol specification. This confidence comes from a belief that the protocol is properly implemented and that it is unlikely that the hosting server can be compromised. On the other hand, and in the absense of further information, our degree of confidence that an open-access workstation running freeware follows the protocol and/or cannot be compromised, is low as it may be subject to a variety of attacks ranging from Trojan Horses in the protocol implementation to vulnerabilities such as buffer overflows in the underlying system.

**Example 3** The components in Example 1 are defined with the following intervals.

| $c$ | $int(c)$ | $rating(c)$ |
|---|---|---|
| $A$ | [public, Ka] | med |
| $B$ | [Kab, Kb] | med |
| $T$ | [Kab, $\top$] | hi |

Principal $A$ manages keys in the range [public, Ka], reflecting that it sends/encrypts data on the public channel and decrypts/receives up to Ka class information. Note that in this deployment we assume that principals $B$ and $T$ do *not* write to the public channel.

The table also provides sample confidence *rating* for principals. We have medium confidence that principal $B$ properly protects its key information in the range [Kab, Kb]—this assumes that $B$ does not access to the public channel—perhaps $B$ is known to be confined to a protection domain which does not give it direct access to the public channel. In practice, the rating of a component should depend on the keys it protects: we might have a high degree of confidence that $B$ does not write Kb information to the public channel while have a medium degree of confidence that it does not write Kb to Kab. For the sake of simplicity in this paper we restrict ourselves to the interpretation of the *rating* function as a property of the component that is independent of the keys it manages. $\triangle$

**Definition 1** Each protocol component must meet the minimum required confidence, that is, for every component $c$ then

$$\forall x, y : \mathcal{K} \mid int_{\bot}(c) \leq x \leq int_{\top}(c) \wedge int_{\bot}(c) \leq y \leq int_{\top}(c) \\ \Rightarrow minConf(x, y) \leq rating(c)$$

that is, the component achieves the required degree of confidence for every pair of keys that it manages. $\triangle$

When a deployed protocol executes, there is a resulting flow of messages between components. Let $A_x \rightsquigarrow B_y$ represent a flow of a message encrypted using an $x$-level key by $A$ and decrypted using a $y$-level key by $B$.

**Definition 2** A protocol configuration is *classification-safe* if for all components $A$ and $B$ then,

$$A_x \rightsquigarrow B_y \Rightarrow x \leq y \wedge int_{\perp}(A) \leq x \leq int_{\top}(A) \wedge int_{\perp}(B) \leq y \leq int_{\top}(B)$$

$\triangle$

**Example 4** The protocol in Example 1 has direct flows $A_{\mathsf{public}} \rightsquigarrow T_{\mathsf{public}}$, $T_{\mathsf{Ka}} \rightsquigarrow A_{\mathsf{Ka}}$ and $T_{\mathsf{Kb}} \rightsquigarrow B_{\mathsf{Kb}}$, and is classification-safe. We expect that the analysis in protocol entailment, described in [1], can be adapted to provide a semantics for the $\rightsquigarrow$ relation. $\triangle$

## 3 Cascading Risks in Protocols

The *cascade vulnerability problem* [9, 8] is concerned with secure interoperation, and considers the *assurance risk* of composing multilevel secure systems that are evaluated to different levels of assurance according to the criteria specified in [9]. The transitivity of the multilevel security policy upheld across all secure systems ensures that their multilevel composition is secure; however, interoperability and data sharing between systems may increase the risk of compromise beyond that accepted by the assurance level. For example, it may be an acceptable risk to store only secret and top-secret data on a medium assurance system, and only classified and secret data on another medium assurance system; classified and top-secret data may be stored simultaneously only on 'high' assurance systems. However, if these medium assurance systems interoperate at classification secret, then the acceptable risk of compromise is no longer adequate as there is an unacceptable cascading risk from top-secret across the network to classified. Similar cascading risks can be demonstrated in a security protocol.

**Example 5** We extend the protocol in Example 1 to include mutual authentication using the session key $K_{AB}$.

$$\begin{aligned} &\text{Msg4 } B \rightarrow A \ \{\!|A, B, T_T|\!\}_{K_{AB}} \\ &\text{Msg5 } A \rightarrow B \ \{\!|A, B, T_T - 1|\!\}_{K_{AB}} \end{aligned}$$

As a consequence of these additional protocol steps we have additional explicit flows between $A$ and $B$ involving the key $K_{AB}$ in the protocol. These flows ($A_{\mathsf{Kab}} \rightsquigarrow B_{\mathsf{Kab}}$ and $B_{\mathsf{Kab}} \rightsquigarrow A_{\mathsf{Kab}}$) represent the encryption-sending and receiving-decryption using $K_{AB}$ by both $A$ and $B$. Since $A$ and $B$ are trusted to manage this classification of session key, these flows are classification-safe.

The confidence rules require a minimum confidence level of $\mathsf{hi}$ for a component to be considered trusted to manage both $\mathsf{Ka}$ and $\mathsf{Kb}$ keys. Having a confidence of $\mathsf{hi}$, component $T$ is considered trusted to manage both $\mathsf{Ka}$ and $\mathsf{Kb}$ keys. Confidence can be considered to represent the degree of confidence that one can have that a component cannot be compromised. In this case, the effort required by an

attacker corresponds to the effort required to compromise the hi-rated $T$, and, for example, reveal $K_A$ to $B$ using the following (modified) message run.

$$\alpha\text{Msg 2: } T \rightarrow A \ \{|K_{AB}, A, B, N_A, T_T|\}_{K_A}, \{|K_{AB}, A, B, K_A|\}_{K_B}$$

As long as the effort required to compromise $T$ is at least hi, then the risk of this attack is considered acceptable.

Consider an attacker that compromises component $A$. In this case the effort required by the attacker corresponds to the effort to compromise a med rated component $A$, and, for example, embed a Ka classified key in a message encrypted by session key with classificiation Kab, and send it to $B$; this effectively copies a Ka key into a Kb key.

$$\beta\text{Msg } 4 : B \rightarrow A \ \{|A, B, T_T|\}_{K_{AB}}$$
$$\beta\text{Msg } 4 : A \rightarrow B \ \{|A, B, K_A|\}_{K_{AB}}$$

However, the confidence requirement is that in order to copy Ka to Kb requires at least the effort to compromise a level hi rated entity, while the above attack is achieved by compromising med-rated components.

The protocol contains a cascade vulnerability. Individual components meet the minimum ratings based on the $minConf$ confidence rule defined above, however, the interoperation of $A$ and $B$ due to the mutual authentication step in the protocol does not. There is a cascading path from a Ka key managed by component $A$ to a Kb key managed by component $B$, via session key $K_{AB}$. $\triangle$

**Example 6** The previous example assumes that components $A$ and $B$ are trusted to properly manage both long-term and session keys and that these keys are continuously available. However, in practice, both keys are typically not continously available to the component. For example, a Kerberos login client discards its long term key (user passwords) once a ticket/session key has been obtained; if one-time passwords are used, the client does not have any access to the underlying long term secret.

In order to better reflect this situation we model each protocol client ($A$, $B$) in terms of two seperate component entities: a login/connection manager and a session manager. These are intended to correspond to the software components that implement the component. The login manager is responsible for properly using the client's long-term key to obtain the session key and then handing this key off to the client's session manager.

We extend the protocol description to include these components, where $A^{conn}$ and $A^{sess}$ correspond to the connection and session managers that represent $A$, and similarly for $B$.

$$\begin{array}{lll}
\text{Msg 1:} & A^{conn} \rightarrow T & A, B, N_A \\
\text{Msg 2:} & T \rightarrow A^{conn} & \{|K_{AB}, A, B, N_A, T_T|\}_{K_A}, \{|K_{AB}, A, B, T_T|\}_{K_B} \\
\text{Msg 2i:} & A^{conn} \Rightarrow A^{sess} & K_{AB}, A, B, N_A, T_T \\
\text{Msg 3:} & A^{conn} \rightarrow B^{conn} & \{|K_{AB}, A, B, T_T|\}_{K_B}
\end{array}$$

$$\begin{aligned}
\text{Msg 3i: } & B^{conn} \Rightarrow B^{sess} \;\; K_{AB}, A, B, T_T \\
\text{Msg4: } & B^{sess} \rightarrow A^{sess} \;\; \{\!|A, B, T_T|\!\}_{K_{AB}} \\
\text{Msg5: } & A^{sess} \rightarrow B^{sess} \;\; \{\!|A, B, T_T - 1|\!\}_{K_{AB}}
\end{aligned}$$

In this protocol, $P \Rightarrow Q : M$ represents the sending of a message $M$ over an internal/private software channel from $P$ to $Q$ that is assumed secure. In practice this could be implemented as message passing or API calls within the platform that implements the clients.

Figure 2 provides revised interval and ratings for the components of this revised protocol. Note that we assume high confidence for the connection managers

| $c$ | $int(c)$ | $rating(c)$ |
|---|---|---|
| $A^{conn}$ | [Kab, Ka] | hi |
| $A^{sess}$ | [Kab, Kab] | med |
| $B^{conn}$ | [Kab, Kb] | conn |
| $B^{sess}$ | [Kab, Kab] | med |
| $T$ | [Kab, $\top$] | hi |

**Fig. 2.** component intervals and ratings for the extended protocol

on the basis that we have high confidence in the proper operation of the login client and that long-term keys are not available once session keys are issued. The session managers in the example are rated as med but can be safely rated as lo given the minimum requirment $minConf[\mathsf{Kab}, \mathsf{Kab}] = \mathsf{lo}$.

The protocol generates the following flows between its components.

$$\begin{aligned}
&T_{\mathsf{Ka}} \rightsquigarrow A^{conn}_{\mathsf{Ka}};\; A^{conn}_{\mathsf{Kab}} \rightsquigarrow A^{sess}_{\mathsf{Kab}};\; A^{sess}_{\mathsf{Kab}} \rightsquigarrow B^{sess}_{\mathsf{Kab}}; \\
&T_{\mathsf{Kb}} \rightsquigarrow B^{conn}_{\mathsf{Kb}};\; B^{conn}_{\mathsf{Kab}} \rightsquigarrow B^{sess}_{\mathsf{Kab}};\; B^{sess}_{\mathsf{Kab}} \rightsquigarrow A^{sess}_{\mathsf{Kab}}
\end{aligned}$$

Based on the ratings in Figure 2, it follows that the flows between these components are individually classification-safe.

△

## 4 Discussion

In this paper we considered a quantitative-based approach to evaluating the deployment of security protocols. The approach focuses on how confidence in protocol components can be traced across the deployment; it does not analyze vulnerabilities in the underlying behavior of the components (for example API attacks [4]) or their interaction (for example protocol analysis [5, 2]), though such techniques could be used to inform the degree of confidence measure.

A Needham-Schoeder style protocol deployment was analyzed in this framework. For reasons of space and ease of exposition, the deployment was deliberately simplistic. Nevertheless it was possible to demonstrate a variation of the

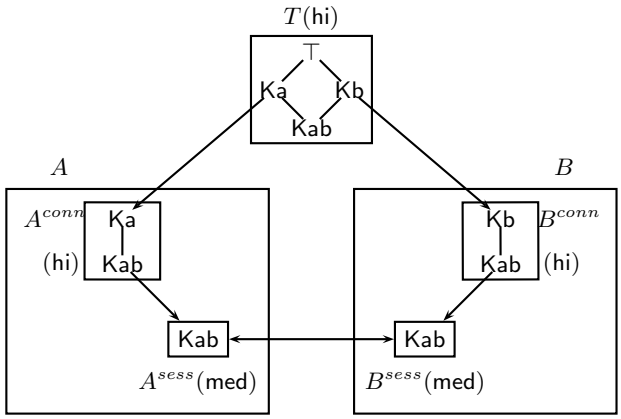


**Fig. 3.** Cascade-free configuration for the Kerberos-style protocol

channel cascade problem in the deployment. It would be interesting to explore a richer deployment that included, for example, authenticator token components and components that are dynamically selected during the initial protocol parameter negotiation phase that is typical of practical protocols.

## References


1. G. Bella and S. Bistarelli. Soft constraint programming to analysing security protocols. *Theory and Practice of Logic Programming*, 4(5):1–28, 2004.
2. Giampaolo Bella. *Formal Correctness of Security Protocols*. Information Security and Cryptography. Springer, 2007.
3. S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume LNCS 2962. Springer, 2004.
4. Mike Bond and Ross Anderson. API-level attacks on embedded systems. *Computer*, 34(10):67–75, 2001.
5. D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
6. S.N. Foley. Conduit cascades and secure synchronization. In *Proceedings of ACM New Security Paradigms Workshop*, 2000.
7. S.N. Foley, S. Bistaelli, B. O'Sullivan, J. Herbert, and G. Swart. Multilevel security and the quality of protection. In *Proceedings of First Workshop on Quality of Protection, Como, Italy*. Springer LNCS, 2005.
8. J.K. Millen and M.W. Schwartz. The cascading problem for interconnected networks. In *4th Aerospace Computer Security Applications Conference*. IEEE CS Press, December 1988.
9. TNI. Trusted computer system evaluation criteria: Trusted network interpretation. Technical report, National Computer Security Center, 1987. Red Book.