

# Trust and Tamper-Proof Software Delivery

Martin Naedele  
ABB Corporate Research  
CH-5405 Baden, Switzerland  
martin.naedele@ch.abb.com

Thomas E. Koch  
ABB Corporate Research  
CH-5405 Baden, Switzerland  
thomas.koch@ch.abb.com

## ABSTRACT

Software engineering today relies to a large extent on acquiring and composing software components and other software-related artifacts from different producers, either at design or at run time. For any user of such artifacts, both as developer and as end-user, the question arises how to ensure that these artifacts are not malicious. Complete inspection of acquired code is, if not impossible, at least impractical and uneconomical for commercial software. The user thus has to trust the code, or rather its supplier and the delivery channel. This paper examines different trust models in the software supply chain and their rationales.

Any trust-based supply chain also requires as prerequisite a tamper-proof distribution channel. Such channels can theoretically be realized using digital signature technology, but some practical and theoretical challenges remain. The paper outlines the challenges and shortcomings of current commercial approaches, proposes some solutions, and suggests areas for further research.

**Categories and Subject Descriptors:** K.6.5 [Computing Milieux]: Security and Protection, H.4 [Information Systems Applications], D.2 [Software Engineering]

**General Terms:** Security

**Keywords:** software development, security, integrity, digital signature.

## 1. INTRODUCTION

### 1.1 The software supply chain

Software engineering practices today rely heavily on explicit or implicit component-based assembly and reuse on different levels of abstraction. The components are to a large extent not developed under the control and supervision of the (end-)user, but have been delivered by external and internal suppliers in the form of operating systems, applications, development frameworks, development libraries, etc.

Thus almost every user of computer systems, including private and commercial end-users, system integrators and software developers, today uses artifacts obtained from others. Such software-related artifacts include applications and components in compiled binary form or as source code, and documentation. Even software providers rely again on other artifacts. Software production forms a supply chain. This is also true for most security and safety critical software applications, such as control systems for critical infrastructure.

### 1.2 The evaluation problem

With such software supply chain, one key question for the acquirer (either as formal role or as actual end-user) of an externally provided software artifact is whether it will cause harm to the system in which it is used.

It would appear that this question is only answerable by detailed inspection of the artifact. Nevertheless, artifacts that are produced and acquired within a supply chain are mostly not thoroughly and exhaustively inspected and tested for malicious functionality or content by the acquirer. This is on the one hand due to reasons of cost and effort which are in most cases prohibitive to such exhaustive verification from a practical point of view, and on the other hand due to theoretical problems with actually conducting meaningful inspection.

Practical experience gained over years with evaluation according to the Common Criteria [13] is witness to both the high effort needed and at the same time the futility of investing this effort, demonstrated by the frequent vulnerability announcements for evaluated and certified systems. Moreover, Thompson shows in [26] the theoretical limits of verification. Also, exhaustive evaluation assumes that the supplier is willing to expose its artifact to detailed inspection. Another theoretically promising research area is proof-carrying code, but the practical applicability of this approach to many commercial software artifacts is still limited by constraints like the need to specify the required safety properties in advance and the size of the proofs [20].

Therefore, the acquirer is usually stuck with unverified and unproven artifacts - artifacts that have not been thoroughly and exhaustively inspected and tested for malicious functionality or content. This is even true with regard to most open-source artifacts. Also, third-party components, even if their functionality has been tested to a certain extent by the acquirer, may usually be regarded as unverified from a security point of view.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SESS'06, May 20–21, 2006, Shanghai, China.  
Copyright 2006 ACM 1-59593-085-X/06/0005 ...\$5.00.

### 1.3 Fundamental approaches

The fact that certainty about the absence of malicious behavior in the acquired artifact can not be obtained leaves the acquirer with two alternatives:

One alternative is to technically limit the scope of functionality, and thus also of malicious functionality, for the artifact in the system. Technically, this is implemented e.g. as process isolation, mandatory access controls, sandboxing, or blocking anomalous system calls via a host-based intrusion detection/prevention systems. This approach is well feasible for certain application scenarios, especially if the unverified code can be kept clearly separate from other, verified, parts of the application, and if it requires only limited privileges. However, experience, for example with sandboxing in the Java applet context, has shown that many application and usage scenarios can not work within the sandbox paradigm.

The other alternative for the acquirer is to trust the supplier of the artifact. Trust with regard to the suitability of the artifact for the intended purpose (which is actually a broader notion than harmlessness) then replaces assurance of harmlessness of the artifact otherwise obtained by inspection. In theory, one might argue that this is a rather weak approach - as Lenin is quoted to have said "trust is good, verification is better" -, but in practice this may very often be the best that can be done in a time and resource constrained environment.

### 1.4 Contributions

Starting from the proposition that trust along the supply chain is a major enabler of today's software economy, this paper makes the following contributions:

Section 2 introduces a framework of trust models to support rational decision making when participating in a trust-based supply chain for software artifacts.

Tamper-proof delivery of software artifacts is a prerequisite for a trust-based software supply chain. In Section 3 this paper outlines a number of issues in implementing tamper-proof delivery that have so far received little attention both in research and from solution providers, and suggests solutions. In particular, the usefulness of certificate expiration is put into perspective.

## 2. TRUST-BASED SOFTWARE DISTRIBUTION

The decision to rely on a trust-based software supply chain as well as the selection of the "trusted" suppliers has to take the reasons for trusting those suppliers into account. The acquirer must match its business and operational environment with the risk incurred by using artifacts from a certain supplier.

While the notion of trust may be intuitive, it is not easy to define precisely. This may be because here technical and social domains meet, as [27] suggests. The most useful definition we found is from [1]:

Trust: In cryptology and crypto systems, that characteristic allowing one entity to assume that a second entity will behave exactly as the first entity expects.

Note that this definition makes no objective statement about the benevolence or maliciousness of the trusted entity.

This section introduces "trust models" as a means to make this risk explicit and to support rational reasoning about it.

### 2.1 Trust models

The trust the acquirer places in an artifact is in fact (only) a trust in the suppliers of the artifact along its supply chain.

This trust can be based on different types of rationales, which shall be called "trust models" in the following. Such trust models are:

- **Liability:** The supplier would be liable for any damages caused by its harmful artifacts. The prospective of successfully claiming compensation for damages from the supplier may be a sufficient risk mitigation argument for the acquirer.
- **Reputation:** The supplier's reputation would be severely damaged if it became known that it was responsible for harmful artifacts. The damage may refer to an individual developer's self-esteem or to the ability of a company to continue to do business. Reputation is typically a big issue for companies with a business model based on being trusted, such as commercial certification authorities.
- **Strong interest:** Doing harm would be against the interest of the supplier of the artifact beyond reputation and liability concerns. The supplier could e.g. have a particular interest that the specific application in which the acquirer uses the artifact succeeds, e.g. because they are part of the same organization, or because this application is its source of revenue.
- **Weak interest:** A weaker variant of the interest trust model is just to assume that the supplier has no active interest in harming the acquirer.
- **Proven in use:** The artifact has been in wide-spread use for considerable time and no harmful effects have been observed so far. This type of trust model is used to justify reliability assumptions within the functional safety community [12]. However, in the safety context the adversary is nature (the laws of physics), whereas in security it is an intentionally malicious human. For nature, assumptions can be made that allow valid statistical treatment. In the security context the harm may be intentionally hidden (covert channel) or limited to certain trigger conditions (e.g. date). The proven-in-use trust model is thus not as reliably applicable with regard to trust in the security context.
- **Directive:** Trust in the artifact is ordered by an organizational instance that is entitled to make such prescriptions. An example are end-users within an enterprise that are instructed to trust applications provided by the central IT unit of that enterprise.
- **Idealism:** Producing harmful artifacts is believed to be against the moral principles of the supplier.
- **Blind:** The artifact is trusted to be harmless even though there is no specific rationale or evidence to substantiate this belief. Blind trust is common, especially for home users, and may very well be a reasonable and appropriate trust model if there is only little value at stake.

These trust models extend the three trust foundations proposed in [27]. That paper proposes another trust foundation "trust based on policy enforcement", but we feel that this concept does not quite fit into the same category as the other trust models, as it strives to remove the fundamental element of uncertainty otherwise inherent in trust models.

[6] suggests to have trust in well-known vendors without further explaining whether this suggestion is based on what is called here the interest, reputation, or liability trust model.

Obviously, not all trust models are equally strong and applicable for the same situations. The risk analysis of the acquirer has to take the trust model into account. The acquirer might even combine the trust based approach with some of the other functionality limiting means mentioned in Section 1.3 [25].

## 2.2 Trust along the supply chain

In all but the most trivial situations the party recognized as the supplier (a person or an organization, depending on the considered level of abstraction) will not have created the supplied artifacts all by itself. Thus it is likely that it again will have to trust other parties, such as sub-suppliers, developers, builders, or testers, inside and outside the supplier organization.

The decision of the acquirer on the trust model to be used may thus depend not only on its direct supplier, but also on the trust models and supporting mechanisms and processes the supplier uses towards its internal and external sub-suppliers.

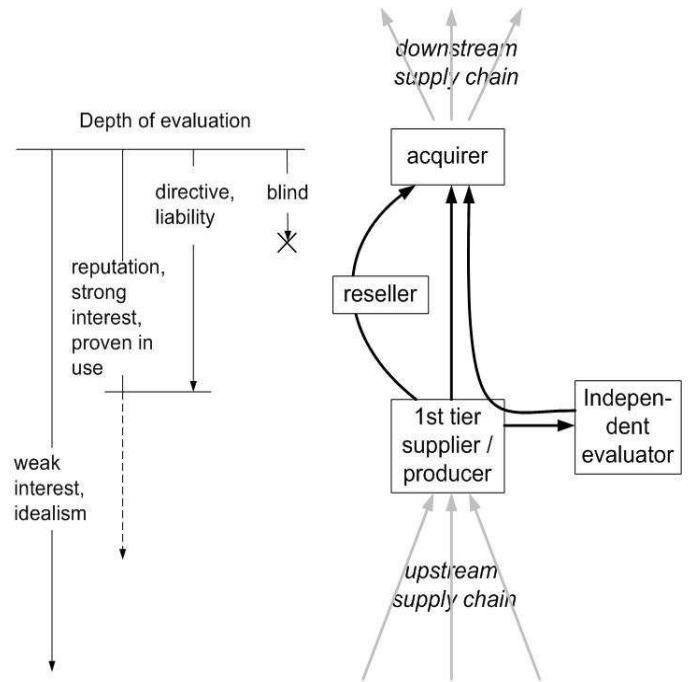
Figure 1 shows how trust models are applied along the artifact supply chain. The supply chain will, of course, in realistic cases likely be a tree, as multiple inputs are integrated into an artifact. To establish the integrity of an artifact, the acquirer may, depending on the trust models used, have to evaluate this chain or tree back for several stages until a supplier relationship is found with a trust model that provides the desired degree of risk mitigation. Two special cases are also shown in the figure. A reseller that does not terminate the tamper-proof channel is just a part of the delivery logistics, but not a part of the logical trust chain. The independent evaluator may be considered as a supplier with respect to the trust model decision, if the evaluator actually signs the evaluated artifact so that the acquirer can be sure that it obtains exactly the evaluated artifact, even though the actual delivery is effected by e.g. the supplier/producer.

For the following trust models it suffices to limit the evaluation to the 1st tier supplier:

- Trust by directive, because the directive is, by definition, not questioned.
- Liability, if and because the supplier is liable for all included components.

For the following trust models the evaluation may end at the 1st tier supplier:

- Reputation, because the supplier reputation may or may not cover all included components. It will likely not cover a component that is resold packaged, but not integrated, and is a well-known artifact by itself (e.g. an operating system). This trust model is also likely not sufficient for suppliers or independent evaluators with regard to artifacts which are known to



**Figure 1: The trust-based supply chain and the necessary depth of evaluation depending on the trust model used.**

be potentially harmful or if the supplier's inspection practices are known to provide only weak assurance of harmlessness, such as the lower Evaluation Assurance Levels (EAL) according to the Common Criteria (ISO/IEC15408) [23].

- Strong interest, because the strong interest of the supplier in supplying harmless artifacts may or may not cover all included components (see also reputation).
- Proven in use, because in order to establish the exact scope of the successful previous usage of the artifact it may be required to evaluate subcomponents.

For these trust models the evaluation should not end at the 1st tier supplier:

- Weak interest: That the supplier itself has no active interest in harming the acquirer does not mean it will actively take measures to prevent such harm from its suppliers.
- Idealism: That the supplier itself strives not to do any harm does not guarantee it will actively and successfully take measures to prevent such harm from its suppliers.

The need to follow the upstream supply chain (or tree) to all leaves reduces these last trust models in practice to blind trust, for which no evaluation occurs by definition.

## 2.3 Tamper-proof delivery as prerequisite for trust-based distribution

A trust-based approach to managing the risk of software artifacts relies, at least implicitly, on the pre-condition that the actual supplier of the artifact to the acquirer is the same

entity that the acquirer assumes as basis for his trust model and that the artifact has not been modified in transit. This is called "trusted distribution" in [2].

In practice, a trust-based risk management approach thus requires technical means to address two issues along the supply chain: source authenticity and integrity.

- **Source authenticity** is the property of the artifact to actually originate from the source entity the acquirer believes it to originate from. For this purpose it has to be ensured that an attacker can not successfully spoof a (trusted) supplier.
- **Integrity** is the property of the artifact to reach the acquirer in exactly the same state in which it left the supplier entity assumed for trust-based risk mitigation. This supplier entity may be the original creator or a trusted intermediary such as a reputable vendor or a certifying institution. For the purpose of integrity it has to be ensured that the artifact has not been tampered with in transit from the trusted supplier to the acquirer. In practice, tampering can generally not be prevented but it must at least be possible to detect such tampering and reject the artifact in such case. For electronic artifacts it has to be pointed out that "in transit" covers not just the time it travels on some communication network, but refers to the whole time and all the artifact locations between the time the trusted supplier last exerted control over it and the moment the acquirer uses it or at least establishes control over it.

Both these properties, source authentication and integrity, always need to be considered together. As soon as one of them cannot be established, the other becomes irrelevant as well.

Cryptography provides technical means to address these issues via digital signatures and certificates [22, 17, 19]. However, equally important as the security of the employed cryptographic mechanisms is to design and implement the whole process for tamper-proof deployment of artifacts in such a way that an attacker can not circumvent the cryptographic mechanisms.

## 2.4 Previous work

So far, most of the work relating to trust and software artifacts has been done in the area of mobile code [27, 3, 15] and e-commerce [25, 28]. Beyond some white papers (e.g. [8, 4]) from various suppliers of technical solutions for digital signatures and an overview of code signing technologies [10], little has been published on artifact integrity along the software supply chain.

A survey and comparison of several code security approaches and technologies can e.g. be found in [24]. Amoroso et. al. [2] discuss how certain properties of the software development process can be regarded as indicators for the trustworthiness of the resulting artifact. The existence of a scheme for tamper-proof distribution of the software is one such property. Several dangers in the software supply chain, with particular emphasis on open software, are discussed [16].

This paper is not concerned with the threat of tampering with artifacts once they have reached the intended receiver. Such attack scenarios are commonly considered in mobile

agent systems and digital rights management approaches [21].

## 3. CHALLENGES IN TAMPER-PROOF DELIVERY

White papers of digital code signing technology vendors make it seem as if industrial strength code signing was as easy as buying a certificate from a commercial certificate authority and invoking the signing tools of the targeted deployment environment.

However, there are a number of non-trivial issues those papers usually gloss over, that have to be addressed, especially if code signing is relied on to ensure that complex applications for use in critical environments such as industrial automation systems [7] have not been tampered with.

Some of these issues, such as key distribution and revocation checking, are well known and active research topics [9]. Other issues of practical importance have received less attention so far. This section highlights some of the challenges for which definite solutions are still lacking.

### 3.1 Trusted party

The acquirer has a certain mental model of the party it trusts in the context of a trust model. In this mental model, the supplier may be an individual person, e.g. for some open source project, or it may be a legal entity or brand such as a software vendor. The challenge is to represent the entity the mental model refers to in technically usable digital evidence such as a certificate:

Assume there is well-known figure "John Doe" in the open source software community. His work products are known to be of good quality and his views on political and ethical issues are well known through his writings and interviews. Based on this mental model one might trust artifacts from "John Doe" based on "proven in use" or "idealism" trust models. However, how can a link be made from this abstract person to e.g. the information contained in a digital certificate? Even ignoring the ways one may obtain a fake certificate,

there may be dozens of people with the name "John Doe", who are legally entitled to receive a certificate in that name from a commercial CA.

The problem gets even more complicated when looking at legal entities. This paper's authors' company (brand), for example, consists of dozens of legal entities with more or less different names in many countries around the world. These legal companies, would be the entities that would be able to obtain certificates from a commercial certification authority, but customers are unlikely to know them. The legal entities are orthogonal to both the business units and product lines (sub-brands), which the customer is familiar with. Moreover, responsibility for a product line may frequently be shifted between business units (legal entities) and countries (jurisdictions). Quite a number of product lines are still known in the market under the brand of acquired companies. Other legal entities and product lines have been divested over recent years. Purely based on information contained in a certificate it is in this scenario essentially impossible for an acquirer to validate that a certificate issued by a commercial certification authority is the certificate associated with the brand according to the mental model he builds his trust model on.

Using certificates derived from a corporate root certificate of the supplier instead of certificates from commercial certification authorities reduces the number of additional parties that have to be trusted and allows the supplier to structure its system of certificates and information contained in certificates around concepts, entities, and brands that are more useful to the acquirer than legal company names. The necessary out-of-band channel for the certificate can even be associated with the brand by, e.g., including certificate verification information ("fingerprints") into brand logos or printed advertising and documentation. Of course, it requires additional effort on supplier and acquirer side to deploy the supplier root certificate to all acquirers. This scheme is thus more suited to low volume, high value transactions and repeated transactions within an established business relationship, than to Internet-type e-commerce.

### 3.2 Validation by the end-user

Validating the trusted external party and verifying the integrity and authenticity of an artifact may require non-trivial effort and decisions, e.g. on a suitable trust model, based on risk management. In addition, the management of certificates or keys and their revocation lists from multiple external suppliers is technically complex. Therefore, many organizations may not want their end-users to have to deal with these issues.

A solution would be to establish a tamper-proof supply chain on the acquirer side, where a dedicated unit of the organization verifies any artifact acquired from an external party against the certificate of this external party and then re-signs it with a key internal to the acquirer organization. End-users can then operate using the "directive" trust model with regard to all artifacts by the internal organization.

### 3.3 Signing in the development process

A major design problem for the signing process is whether the person who signs the artifact, especially if he is taking a personal responsibility, has at all a realistic chance of knowing what he signs. Ideally, this would be solved by extending the tamper-proof supply chain to the individual developer taking personal, non-repudiable responsibility for every artifact he checks into the configuration management system. Other signatures could attest to successful code reviews and passing several levels of testing and other quality assurance.

Commercial code signing technology should thus support multiple signatures per artifact, so that every acquirer in the supply chain is enabled to evaluate an authenticated history of the artifact. This would also remove the currently existing danger of race-condition based attacks whenever one signature is replaced by another.

### 3.4 Certificate expiration

Certificates typically have an expiration date. This section investigates and, in consequence, questions the usefulness of certificate expiration as a security measure for typical code signing scenarios.

#### 3.4.1 Security objective of certificate expiration

Common expiration periods of code signing certificates are one or two years<sup>1</sup>. The threat against which certificate expiration is supposed to defend is apparently that of an

<sup>1</sup>See e.g. <http://www.thawte.com/pricing/index.html> or <http://www.instantssl.com/code-signing/code-signing.html>

attacker who is able to brute-force the private key corresponding to the certificate. Certificate expiry is intended to mitigate this threat in that the brute-forcing effort is expected to take longer than the certificate validity period [8].

Another potential rationale for certificate expiration could be that it indirectly limits the damage done by a compromised key. The compromise in this case would include attacks like theft which could be executed faster than brute-forcing. However, with common multi-year expiration periods the achieved risk reduction in this threat scenario would be minimal and thus this rationale for certificate expiration can be disregarded.

#### 3.4.2 Expiration and time-stamping

The major factor determining the effect of certificate expiry on suppliers and acquirers of signed artifacts is whether the signature on the artifact has been time-stamped.

If the signature has not been time-stamped then a signature shall not be verified as valid after expiry of any of the certificates in the associated certificate chain. The supplier will thus be forced to deploy a new version of the artifact with a new signature corresponding to a fresh certificate as well as the certificate itself to all previous acquirers in step with each certificate renewal period [8]. The logistics effort of doing this regularly is typically undesirable for both supplier and acquirer.

If a time-stamp is included in the signature, then even an expired certificate can still be used for signature verification. If the signature time and date indicated in the time-stamp are before the expiration date of the certificate then the verification process succeeds. Of course, the time-stamp has to be signed by the time-stamping service to protect against forged time-stamps.

Time-stamping has a number of theoretical and practical challenges in itself (see e.g. [11, 8]):

1. The time-stamping service has to be available whenever needed.
2. The time-base has to be accurate and protected against tampering.
3. Different time-zones have to be taken into account.
4. The certificates of all involved time-stampers have to be delivered to all acquirers using a trustworthy (out-of-band) channel and the verification tools on the acquirer side have to verify the certificate of the time-stamper.
5. The time-stamping process has to be protected against attacks, including attacks where the time-stamping authority is colluding with an attacker to pre-date artifacts.

Issue 1-4 are engineering considerations while issue 5 is an active research topic. Several schemes to reduce the necessary trust in the time-stamper exist [11, 14, 18, 5]. These are either based on establishing a position in a sequence of signing events by integrating into the time-stamp information about artifacts that were time-stamped before and after the artifact of interest, or on using multiple independent time-stampers. Both approaches have considerable overhead in time-stamp creation and require collaboration of multiple parties for time-stamp verification. They may thus be usable

for documents, which are rarely challenged and for which execution of the verification procedure is a rare event, but not for software artifact signing, where verification occurs regularly and must be executed automatically, efficiently, and in constrained, short time.

### 3.4.3 Effectiveness as security measure

Certificate expiration and time stamping introduce additional complexity and overhead into the code signing process. This effort can only be justified if, in compensation, it achieves a significant improvement in security. Is certificate expiration an effective mechanism against the attack it addresses?

The attack model assumes an attacker who is capable to brute-force the private key associated with the certificate in a time that is longer than the certificate validity period, but still useful to him.

If we assume that the attacker has found the private key, he is then able to modify or forge the artifact and sign it, thus spoofing the certificate owner. If he submits this new document to the time-stamping service, later verification by the acquirer will fail because the time-stamp of the document is past the certificate end date.

However, if one assumes that the attacker has the capability to brute-force the supplier certificate private key, one can equally well assume that the attacker can also brute force the private key of equal strength of the time-stamping authority, investing the same order of magnitude of effort. The search for both keys can proceed concurrently. Once the attacker has found the private key of the time-stamping authority, he can issue himself arbitrarily pre-dated time-stamps and thus circumvent the perceived protection of expiring certificates.

The time-stamping authority can, of course, defend against this attack by again implementing an expiration/time-stamping scheme for its certificate, involving a second time-stamping authority, and so on. Extending this time-stamping chain will be likely impracticable from the point of view of the users - signing supplier and verifying acquirer - before it creates a significant hurdle for the attacker. In consequence, we can assume that the time-stamping certificate does not expire, or at least, that its expiry is not checked by the verifiers by means of an additional time-stamp. At this point, the effort necessary for the attacker to forge an artifact is roughly twice the effort of brute-forcing a private key, assuming the same key length for code signing and time stamping certificate. Obviously, the same protective effect could have been achieved if the supplier had originally signed the artifact twice with different keys.

In fact, as the effort to brute-force two keys of a given key length is on average only twice that of breaking an individual key, it would be a more effective strategy to use one single, but longer, key for signing instead of requiring certificate expiration and time-stamping. This would also have practical advantages, as only one certificate would have to be distributed to acquirers and the communication with a time-stamper would be unnecessary.

The above argument against time-stamping applies to code signing. There are, of course, usage scenarios where time-stamping is important, because a certain creation date, absolute or relative to other artifacts or events, has to be documented. In this case, the time-stamp itself is relevant information, not just an ingredient of the verification procedure [11].

From a practical point of view, it should be noted that in some implementations of code signing and verification technologies either the time-stamp or the certificate of the time-stamper are not actually examined.

## 3.5 Compromise of the signing key

As far as the acquirer is concerned, only a key compromise or expiry of the immediate supplier is of interest. The fate of any earlier link in the chain is, independent of the trust model, without consequences for the acquirer if one can with confidence assume that along the chain the validity of verification credentials (including potential revocation) has always been verified at the moment of signing.

Example: A SW application has been purchased by the enterprise IT organization and has been rolled out in the enterprise. Before the roll out, the IT organization has verified that the crypto-technical means to ensure tamper-proofing between supplier (assuming the trust model ends here) and IT organization were valid. This means that the public key certificate of the supplier had neither expired nor was it revoked. A revocation would have indicated either a compromise of the CA or of the supplier signing key associated with the public key in the certificate. Now the IT organization endorses the application by signing it with their own signing key. From this point on, any end user in the enterprise just has to ensure before (and potentially during) usage that the IT organization certificate/public key used to support his trust model of ordered trust to IT organization are still valid and, of course, that the signature verification succeeds. The supplier at that point may well have gone out of business or have had their signing key or certificate compromised. The IT organization certificate is sufficient for the end user if it can be ensured that no artifacts from the external supplier can reach the end user without being subject to evaluation by the IT organization.

In scenarios using a liability, reputation, or strong interest trust model, the supplier is exposed to an attack from an acquirer after a supplier key compromise. If the acquirer obtains the compromised key, it can forge a harmful artifact and then invoke liability claims or exert other types of pressure on the supplier.

The supplier can protect itself against such attacks by providing non-repudiable proof of the actual artifacts delivered to the acquirer. This can be realized by using an independent, trusted escrow/registration agency or by requiring the acquirer to (counter-)sign a hash of the received artifact.

## 4. CONCLUSIONS

This paper addresses two important issues related to the development of secure software systems in a modern software economy based on division of labor between multiple suppliers and acquirers of software artifacts, namely the issue of trust and the issue of tamper-proof artifact distribution.

While software acquirers often use trust in the supplier as decision criterion to assure themselves that a software artifact is harmless, this reliance on trust and the rationale behind the resulting decisions is often not made explicit.

This paper provides guidance for making more conscious trust decisions by presenting a framework of rationales for trusting a supplier.

Even if the supplier is and deserves to be trusted, trust-based assurance also requires that no attacker can spoof a trusted supplier (source authentication) or modify an ar-

tifact while on transit between a trusted supplier and the acquirer (integrity).

There are technical solutions, such as digital signatures, certificates, PKI, available for providing source authentication and integrity protection for data, but these alone are insufficient for realizing a tamper-proof delivery channel. Additional engineering considerations are necessary, and even the technical mechanisms currently have gaps and weaknesses. Moreover, most of the theoretical work on digital signatures is concerned with documents, where the check of the authenticity is a rare and high-profile event. The proposed algorithms and procedures are thus not directly applicable to code signing, where (semi-)automatic verification of authenticity has to be executed frequently, at low cost, with little effort, and with little time delay.

This paper outlines and suggests solutions to a number of challenges for tamper-proof delivery of software artifacts, namely the association between the mental model about a supplier and its technical representation, artifact validation by the end-user, authenticated artifact history, certificate expiration, and a vulnerability of the supplier if its signing key gets compromised.

The paper also critically investigates the conventional wisdom of using commercial certification authorities for code signing root certificates and of using certificate expiration as a security mechanism in code signing.

## 5. REFERENCES

- [1] Alliance for Telecommunications Industry Solutions (ATIS) Committee T1A1. ATIS Telecom Glossary 2000. <http://www.atis.org/tg2k/>.
- [2] E. Amoroso, T. Nguyen, J. Weiss, J. Watson, P. Lapiska, and T. Starr. Toward an approach to measuring software trust. In *Proceedings., 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 198–218, May 1991.
- [3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, chapter The Role of Trust Management in Distributed Systems Security, pages 185–210. Lecture Notes in Computer Science State-of-the-Art series. Springer-Verlag, 1999.
- [4] K. Brown. Security briefs: Strong names and security in the .net framework. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/strongNames.asp>, December 2003.
- [5] M. de Mare and L. DeCoursey. A survey of the timestamping problem. Technical Report SUNYIT 2004-1, SUNY Institute of Technology Department of Computer Science, 2004.
- [6] P. Devanbu, P. Fong, and S. Stubblebine. Techniques for trusted software engineering. In *Proceedings of the 20th international conference on Software engineering*, April 1998.
- [7] D. Dzung, M. Naedele, T. von Hoff, and M. Crevatin. Security for industrial communication systems. *Proceedings of the IEEE*, 93(6):1152–1177, June 2005.
- [8] M. Ene-Pietrosanu, K. Yiu, A. Crossman, A. Lewis, and D. Murton. Deploying authenticode with cryptographic hardware for secure software publishing. <http://www.microsoft.com/technet/security/topics/cryptographyetc/authenticodets.mspx>, June 2005.
- [9] N. Ferguson and B. Schneier. *Practical Cryptography*. John Wiley & Sons, 2003.
- [10] E. Fleischman. Code signing. *The Internet Protocol Journal*, 5(1):14–26, March 2002.
- [11] S. Haber and W. S. Stornetta. How to time-stamp a digital document. In *Advances in Cryptology - CRYPTO 90 Proceedings*, number 537 in Lecture Notes in Computer Science, pages 437–455. Springer, 1991.
- [12] IEC. Functional safety of electrical / electronic / programmable electronic safety-related systems, part 1-7. Standard IEC 61508, December 1998.
- [13] ISO/IEC. Evaluation Criteria for Information Technology Security, version 2.1. Standard ISO/IEC 15408, December 1999.
- [14] M. Just. Some timestamping protocol failures. In *Proceedings of the Symposium on Network and Distributed Security (NDSS 98)*, March 1998.
- [15] K. Kato and Y. Oyama. *Software Security Theories and Systems*, volume 2609 of *Lecture Notes in Computer Science*, chapter SoftwarePot: An Encapsulated Transferable File System for Secure Software Circulation, pages 112 – 132. Springer-Verlag GmbH, January 2003.
- [16] E. Levy. Poisoning the software supply chain. *EEE Security and Privacy*, 1(3):70–73, May-June 2003.
- [17] W. Mao. *Modern Cryptography*. Prentice Hall, 2004.
- [18] H. Massias, X. Serret, and J. Quisquater. Timestamps: Main issues on their use and implementation, 1999.
- [19] U. Maurer. New approaches to digital evidence. *Proceedings of the IEEE*, 92(6):933–947, June 2004.
- [20] G. Necula. Proof-carrying code. In *Proceedings 24th Annual ACM Symposium on the Principles of Programming languages (PLOP97)*, 1997.
- [21] P.C. van Oorschot. Revisiting software protection. In *6th International Information Security Conference*, volume 2851 of *Springer LNCS*, pages 1–13, October 2003.
- [22] B. Schneier. *Applied Cryptography*. Wiley, 2nd edition, 1996.
- [23] J. S. Shapiro. Understanding the Windows EAL4 evaluation. *EEE Computer*, 36(2):102–105, 2003.
- [24] V. Skoularidou and D. Spinellis. Security architectures for network clients. *Information Management and Computer Security*, 11(2):8491, 2003.
- [25] Y. H. Tan and W. Thoen. Toward a generic model of trust for electronic commerce. *International Journal of Electronic Commerce*, 5(2):61 – 74, 2001.
- [26] K. Thompson. Reflections on trusting trust. *Communication of the ACM*, 27(8):761–763, August 1984.
- [27] U. G. Wilhelm, S. Staamann, and L. Butryn. On the problem of trust in mobile agent systems. In *Internet Society Symposium on Network and Distributed System Security 98*, 1998.
- [28] L. Xiong and L. Liu. A reputation-based trust model for peer-to-peer ecommerce communities. In *Proceedings ACM Conference on Electronic Commerce*, 2003.