Verification Across Intellectual Property Boundaries

Sagar Chaki

HCCS, May 10, 2016

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

REV-03.18.2016.0

DM-0003625

Background & Organization

Joint work with Christian Schallhart & Helmut Veith

- Verification Across Intellectual Property Boundaries. CAV 2007: 82-94
- Verification across Intellectual Property Boundaries. ACM Trans. Softw. Eng. Methodol. 22(2): 15 (2013)
- Verification Across Intellectual Property Boundaries. CoRR abs/cs/0701187 (2007)

Slides from earlier presentations by Christian and Helmut

- Part I: Motivation and Overview
- Part II: Details of the Protocol
- Part III: Conclusions

Verification Across Intellectual Property Boundaries

Part I:
Motivation and Overview

Trust in Verification

Classical verification scenario assumes trust

Software Author

… trusts the verification people don’t leak source code to third parties

Verification Engineer

… trusts he gets to verify actual production code

Realistic scenario ? Works well when software author and verification engineer belong to same organization. How about other cases?

Software-Intensive Technology

Microprocessors

100 billion in use

90% in embedded systems

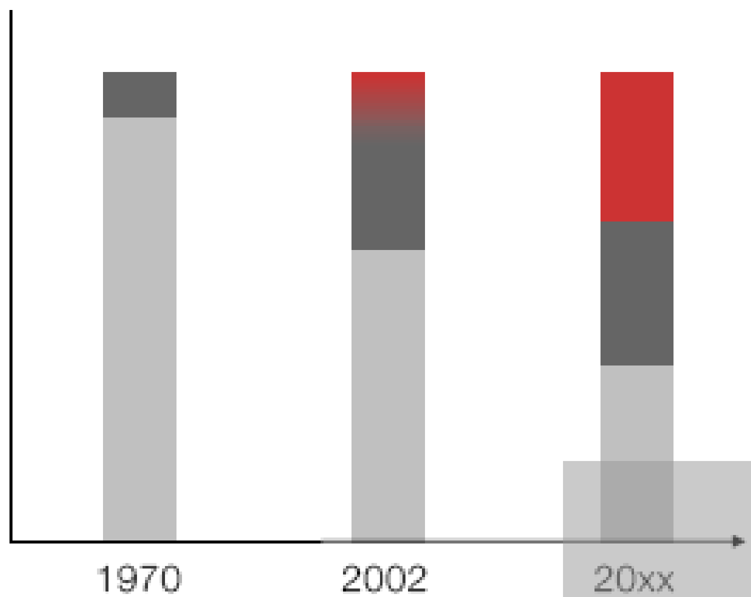
40 in each US household

70 in each BMW 745i

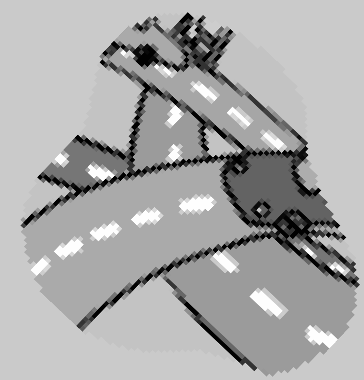
Software Added Value

Added value
in automobile
manufacturing

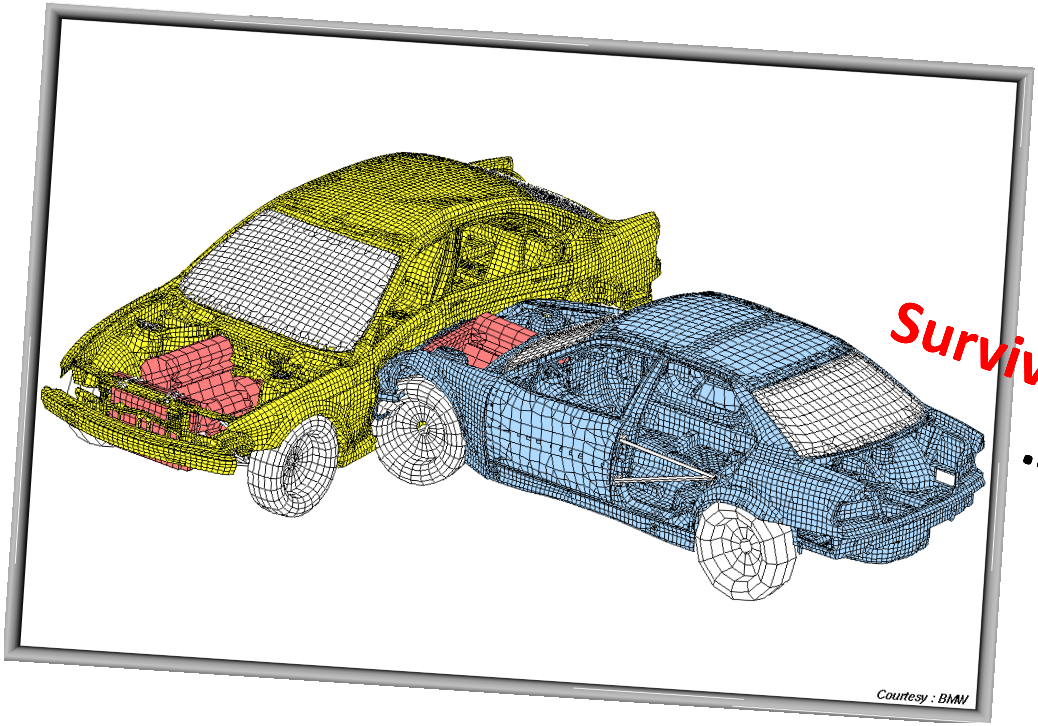
Software
Electronics
Mechanics



Real Time
Safety Critical
Fault Tolerant
Hybrid



Critical Software Quality

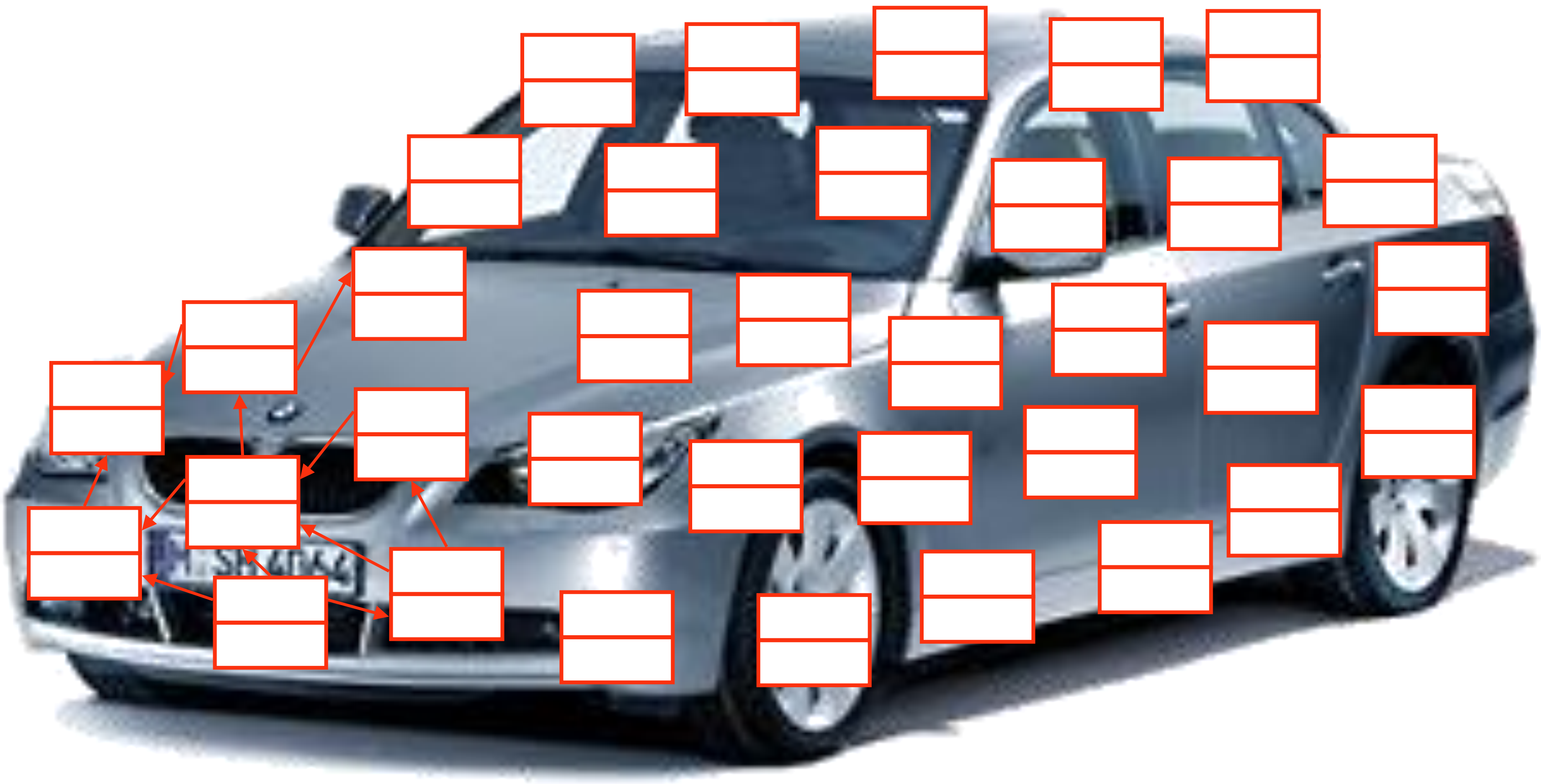


Survival of the passengers

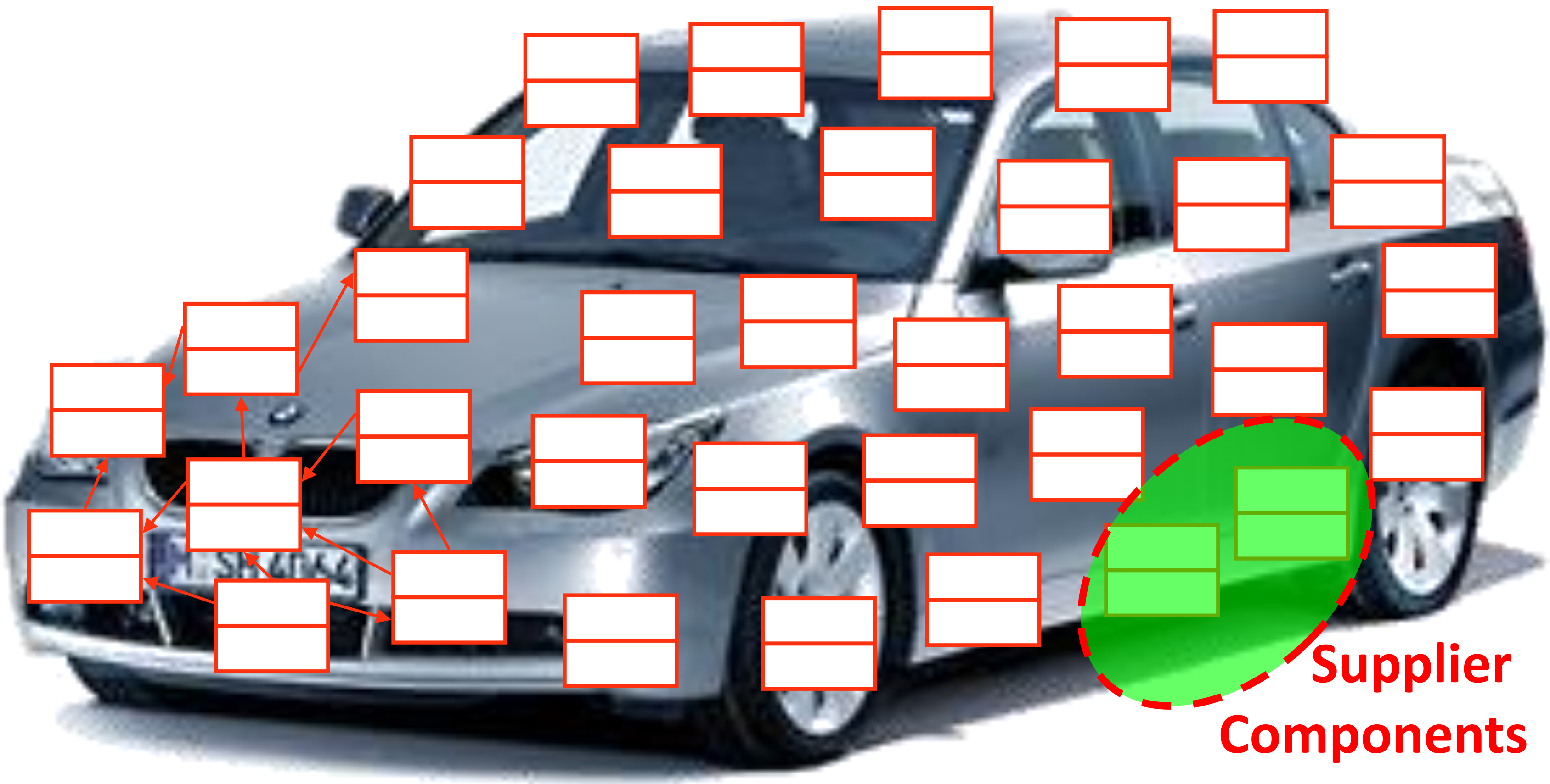
… and the company.

Formal methods and (semi)automated verification.

Assembly from Components
Deep Supply Chains



Verification Barrier



CD player software
engine control
adaptive cruise control

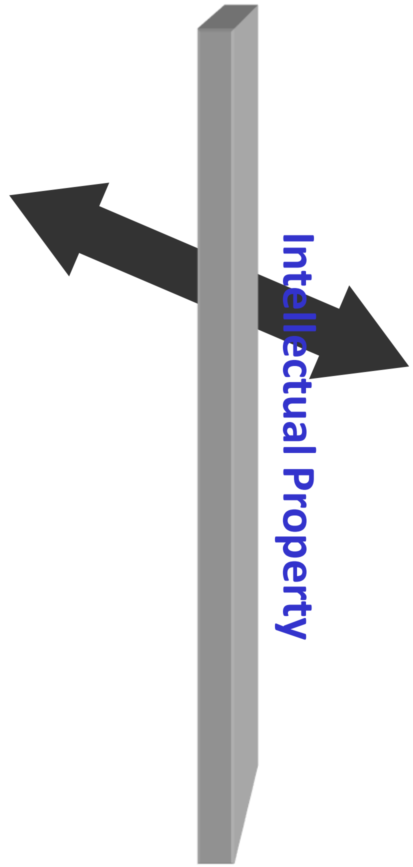
Who verifies suppliers' components ?

Trust in Verification

Classical verification scenario assumes trust

Software Author

… trusts the verification people don't leak source code to third parties



Verification Engineer

… trusts he gets to verify actual production code

Supplier

Car Manufacturer

Microsoft's SLAM

run in kernel mode
written by hardware companies
proprietary source code
error prone

Windows
device drivers

SLAM / SDV helps to find many errors

What is the assurance that developers / companies use SDV in practice ?

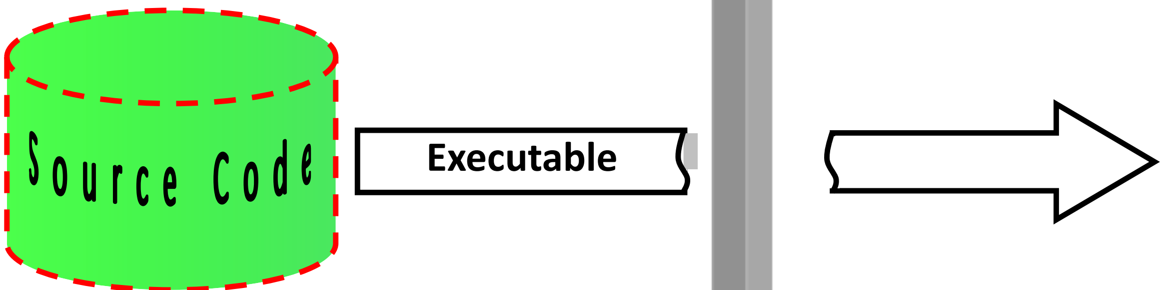
The IP Verification Barrier

Supplier

produces a component

Customer

purchases component



delivers executable

hides source code

receives executable

needs assurance

Verification without revealing the source code ?

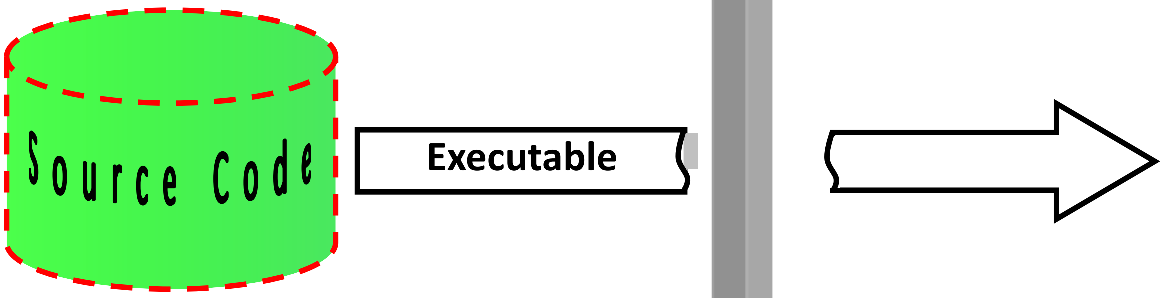
The IP Verification Barrier

Supplier

produces a component

Customer

purchases component



delivers executable

hides source code

receives executable

needs assurance

?

Limited application scope.

Legal Issues.

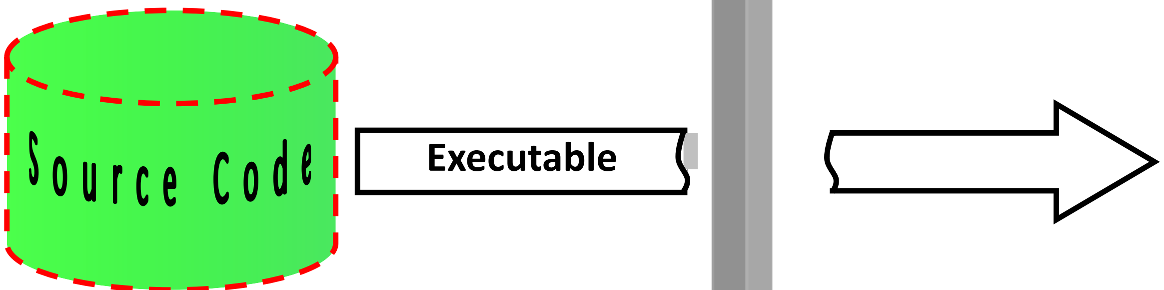
The IP Verification Barrier

Supplier

produces a component

Customer

purchases component



delivers executable

hides source code

receives executable

needs assurance

?

Proof Carrying Code ?

Leaks information about source code.

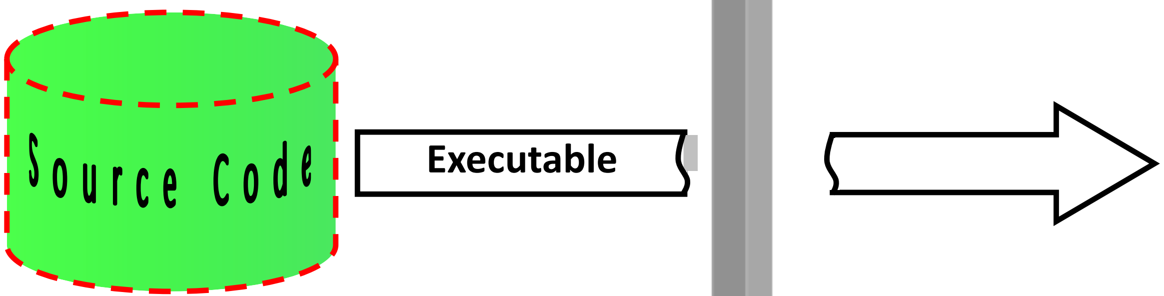
The IP Verification Barrier

Supplier

produces a component

Customer

purchases component



delivers executable

hides source code

receives executable

needs assurance

? **Zero Knowledge Proofs ?**

Proofs leak information.
Require knowledge of source structure.

Restricted to Isabel & cousins.

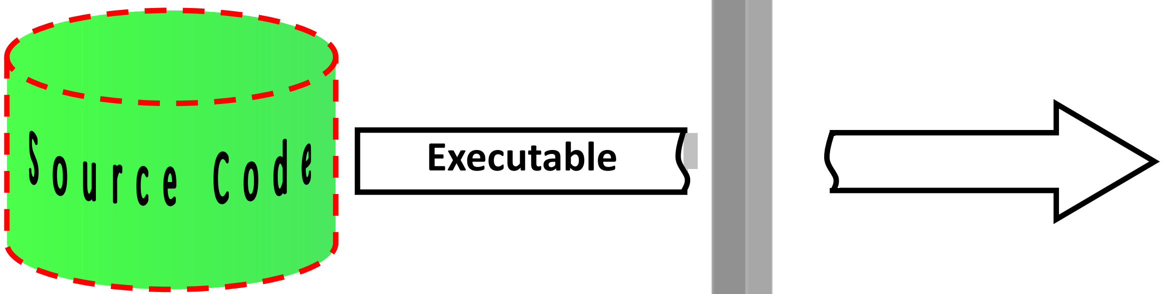
The IP Verification Barrier

Supplier

produces a component

Customer

purchases component



delivers executable

hides source code

receives executable

needs assurance

?

Secure Multiparty Computation ?

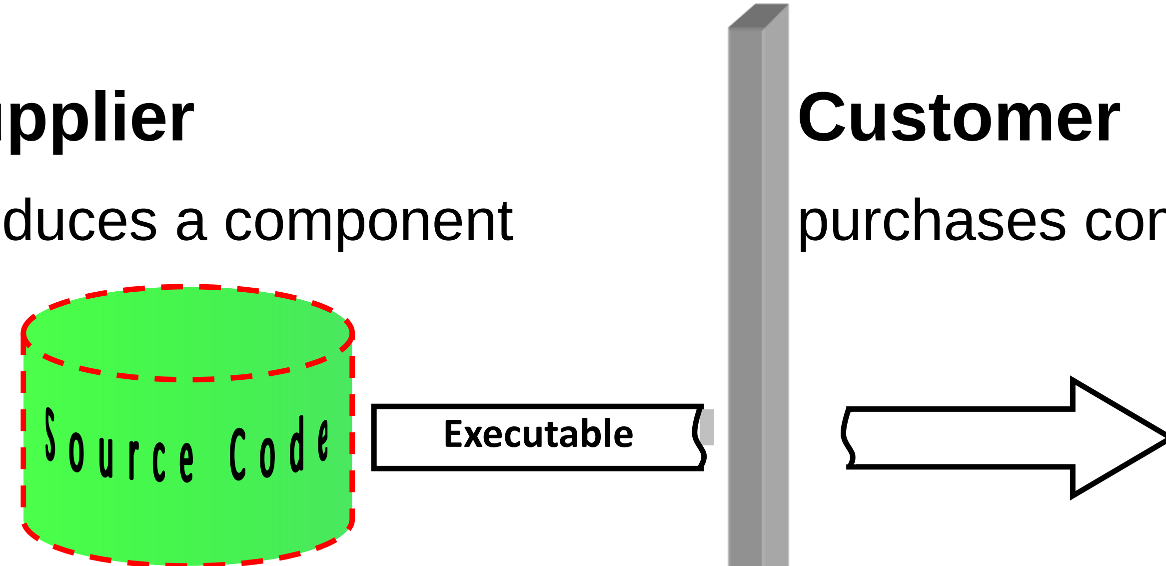
Tailored for single use applications.

Requires transformation of tool chain into circuit.

The IP Verification Barrier

Supplier

produces a component



delivers executable

hides source code

Customer

purchases component

receives executable

needs assurance

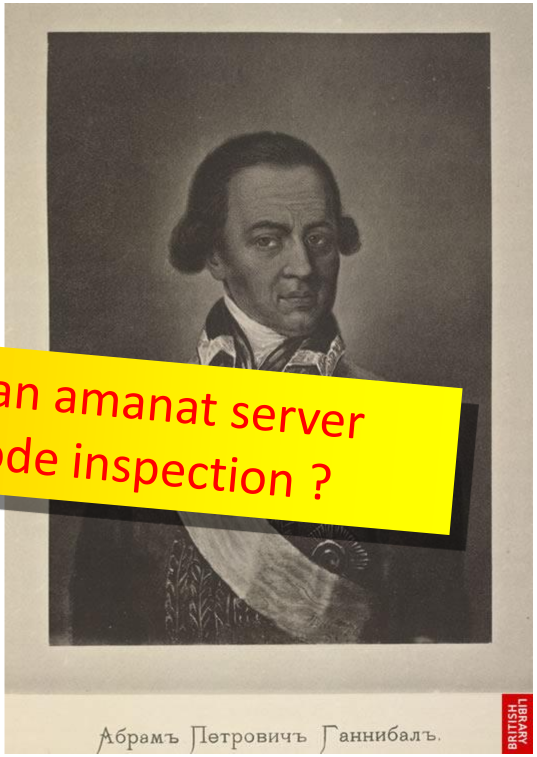
How can we make sure secrecy *after* verification ?

Amanat

ancient judicial term

"*noble prisoner*"

life confined to contract partner's site



Can we use an amanat server for source code inspection ?

Abram Petrovich Gannibal (1696-1781)
Grand²father of A. Pushkin

The IP Verification Barrier

Supplier

produces a component

delivers executable

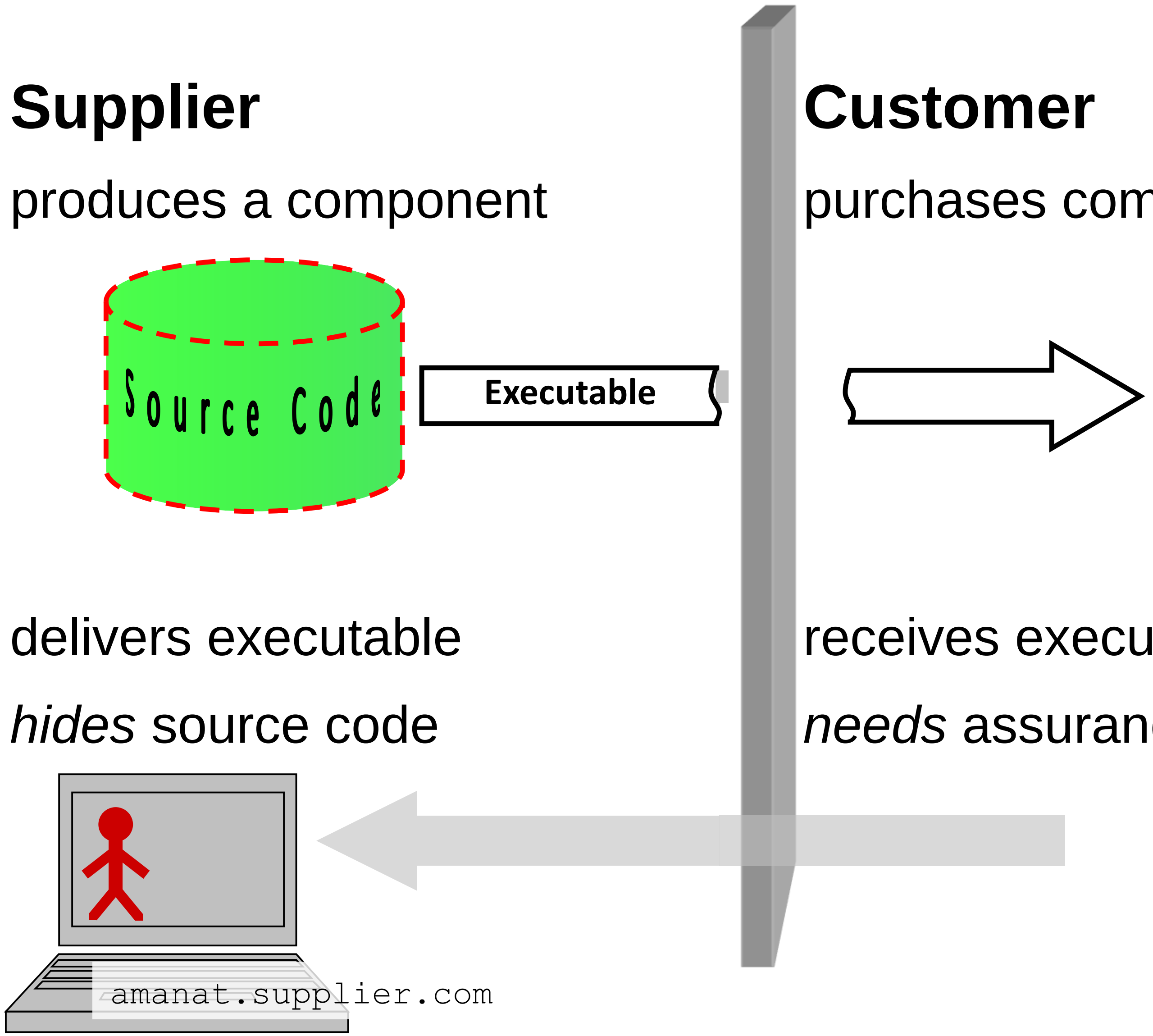
hides source code

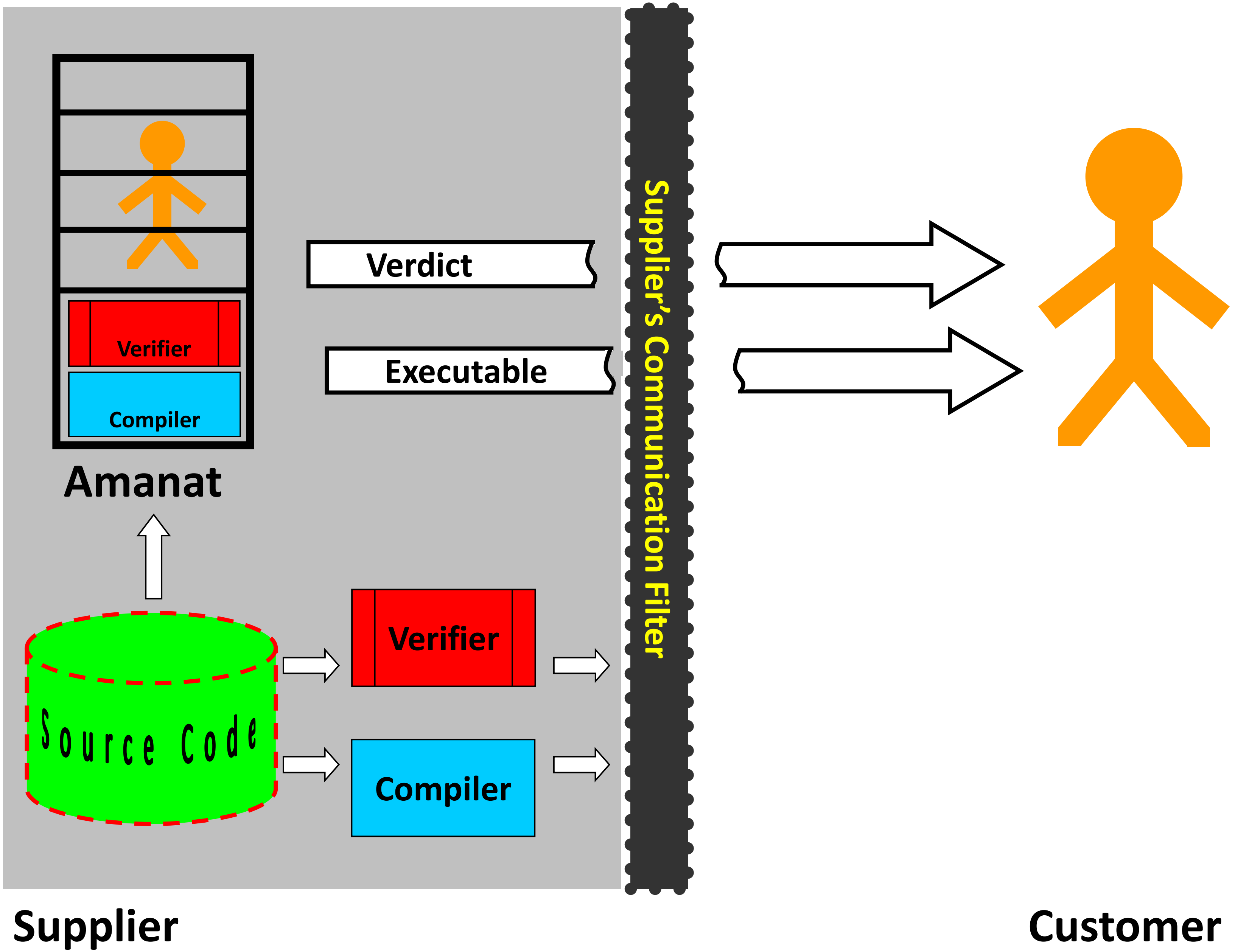
Customer

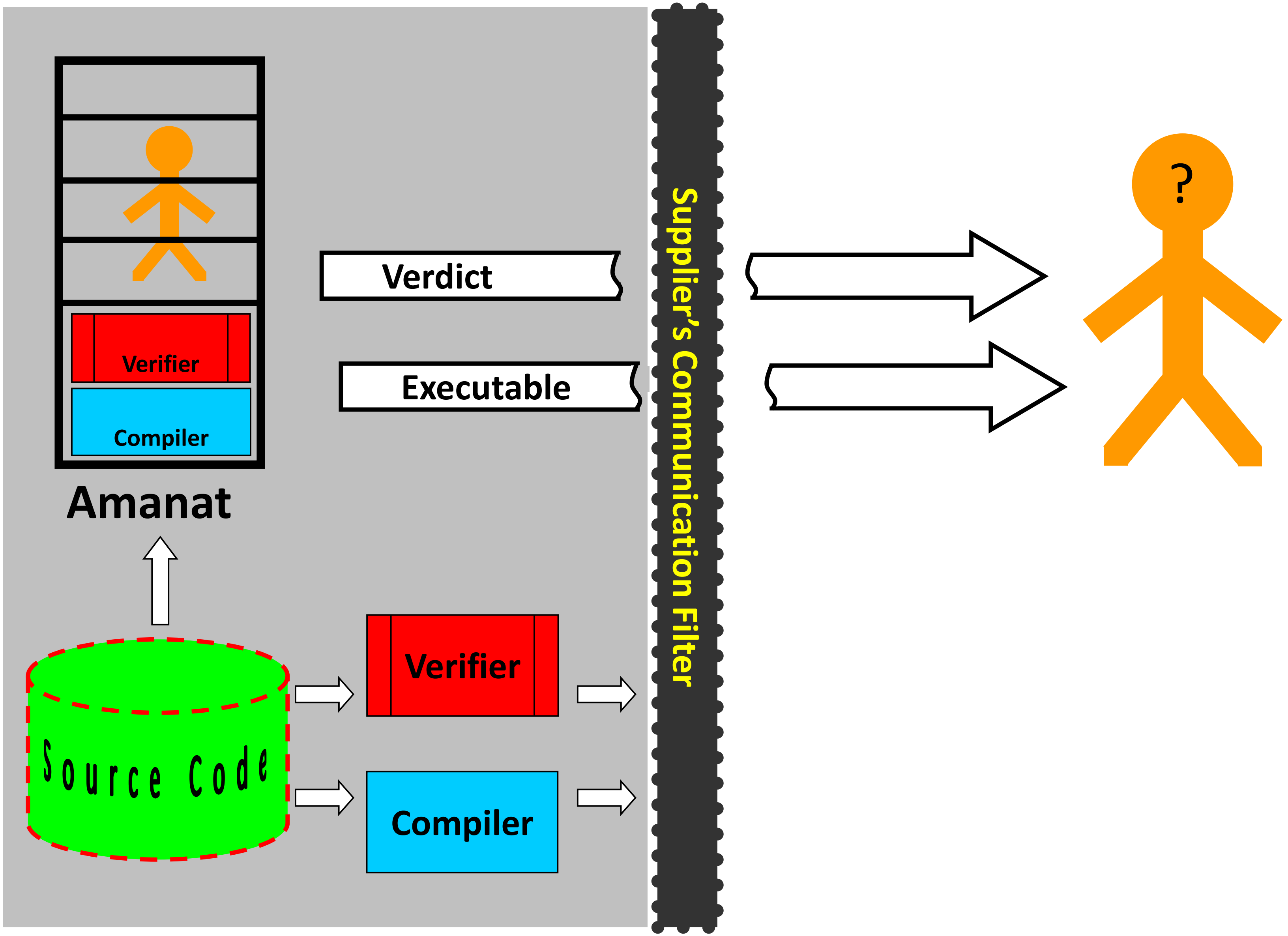
purchases component

receives executable

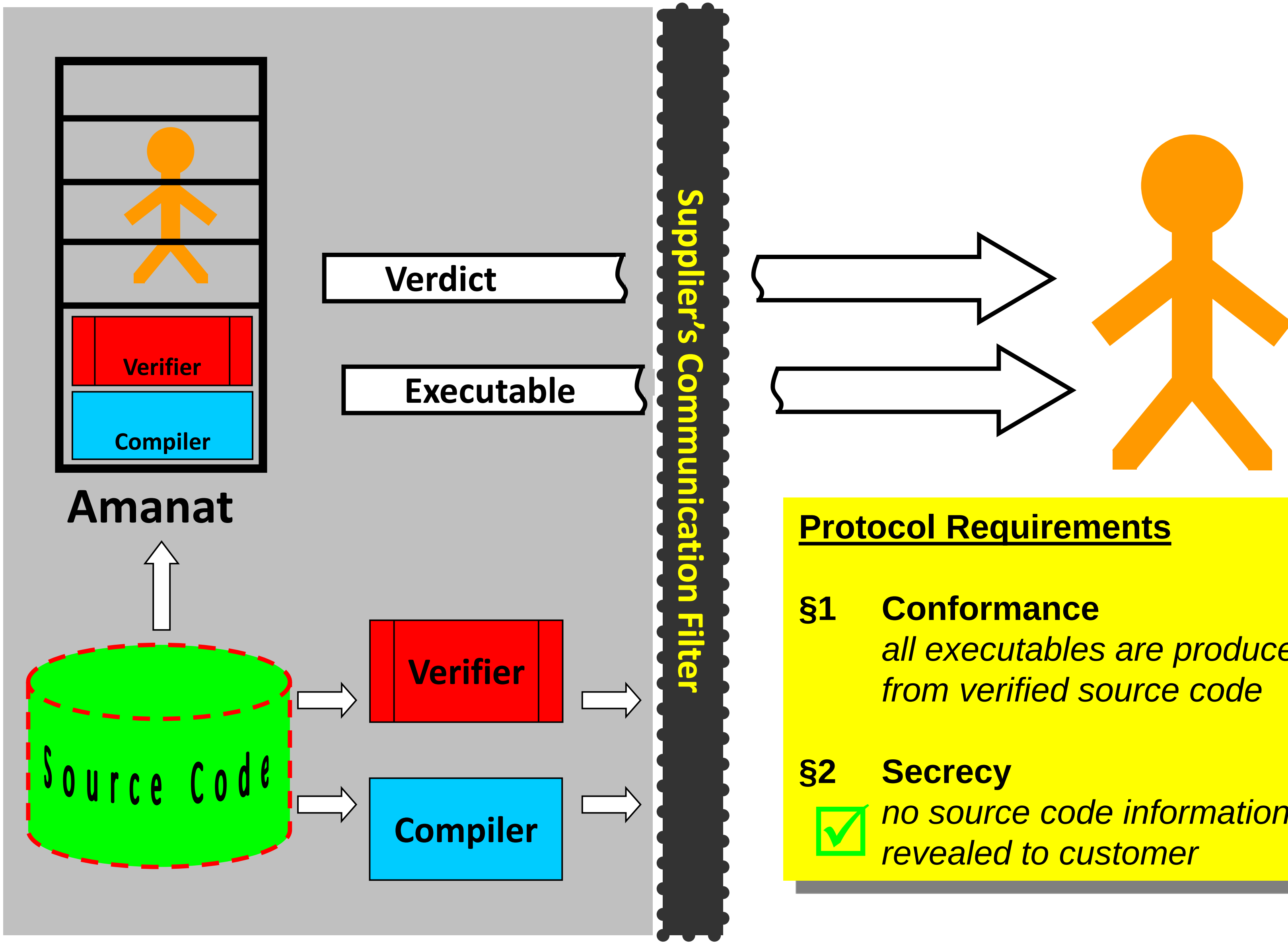
needs assurance




Verdict
Executable
Verifier
Compiler
Amanat
Supplier's Communication Filter
Source Code
Verifier
Compiler
Supplier
Customer



Supplier controls communication channels **Customer**



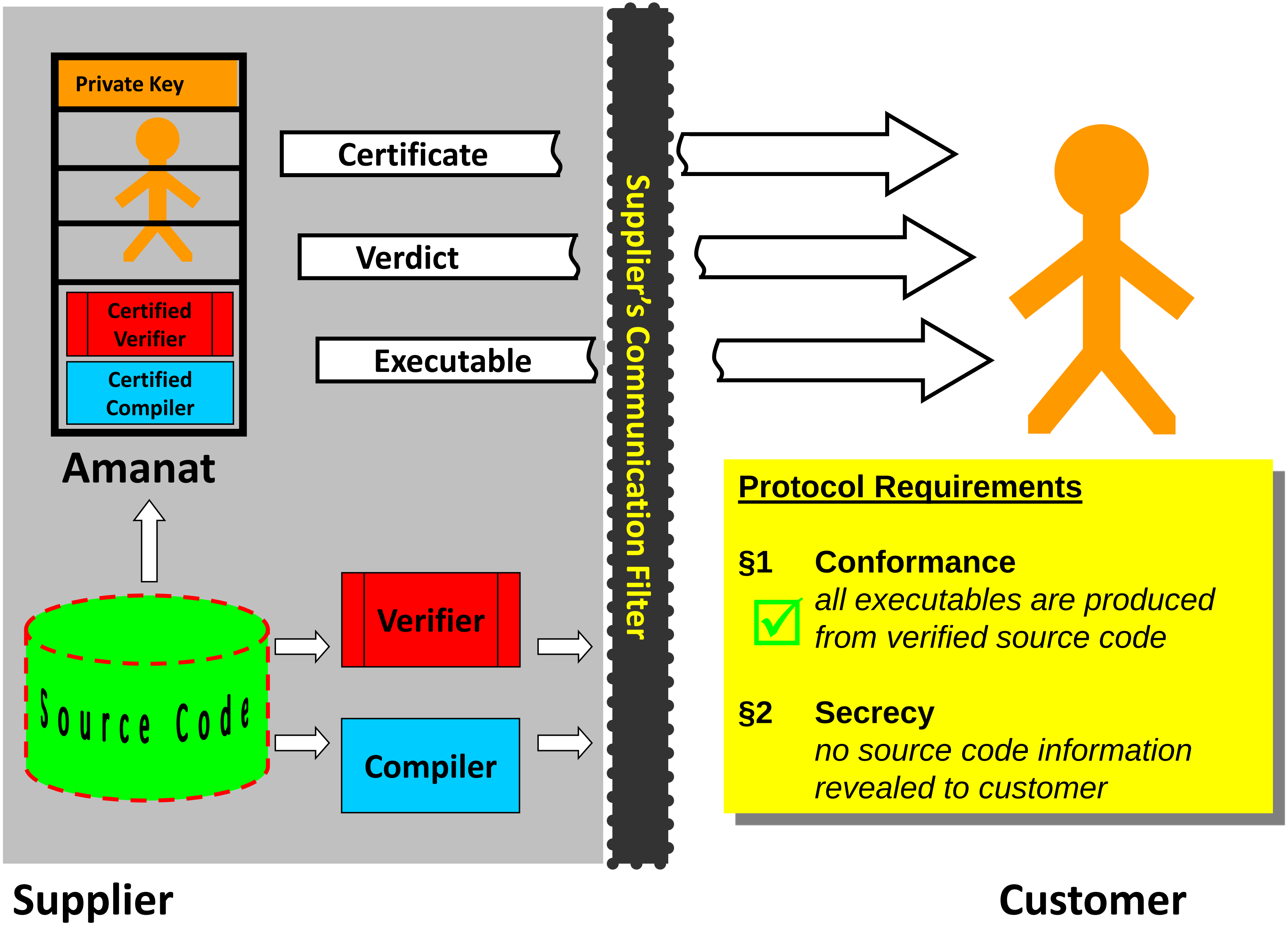
Protocol Requirements

§1 **Conformance**
all executables are produced from verified source code

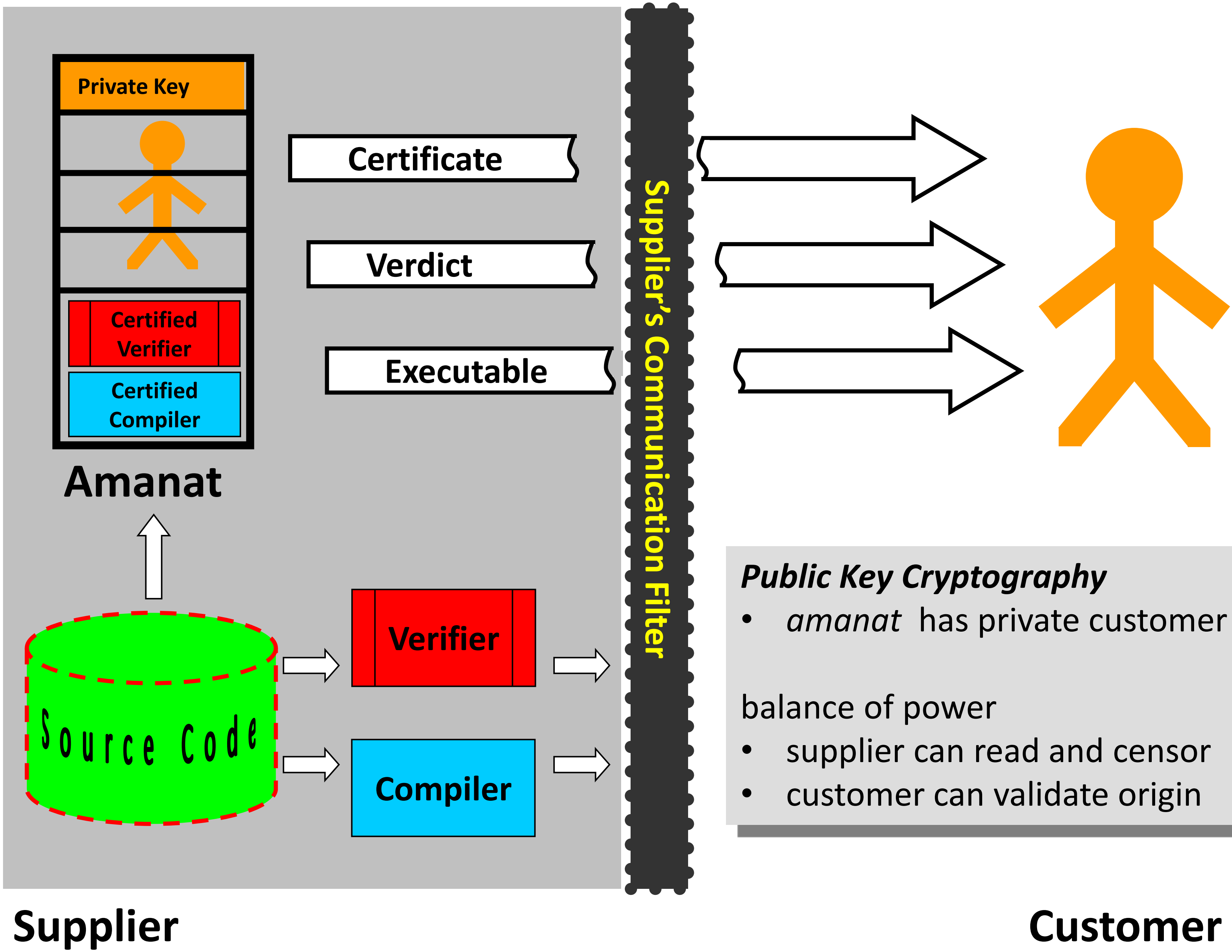
§2 **Secrecy**
✓ *no source code information revealed to customer*

Supplier controls communication channels

Customer



Private Key
Certified Verifier
Certified Compiler
Amanat
Source Code
Verifier
Compiler
Certificate
Verdict
Executable
Supplier's Communication Filter
Protocol Requirements
§1 Conformance
all executables are produced from verified source code
§2 Secrecy
no source code information revealed to customer
Supplier
Customer

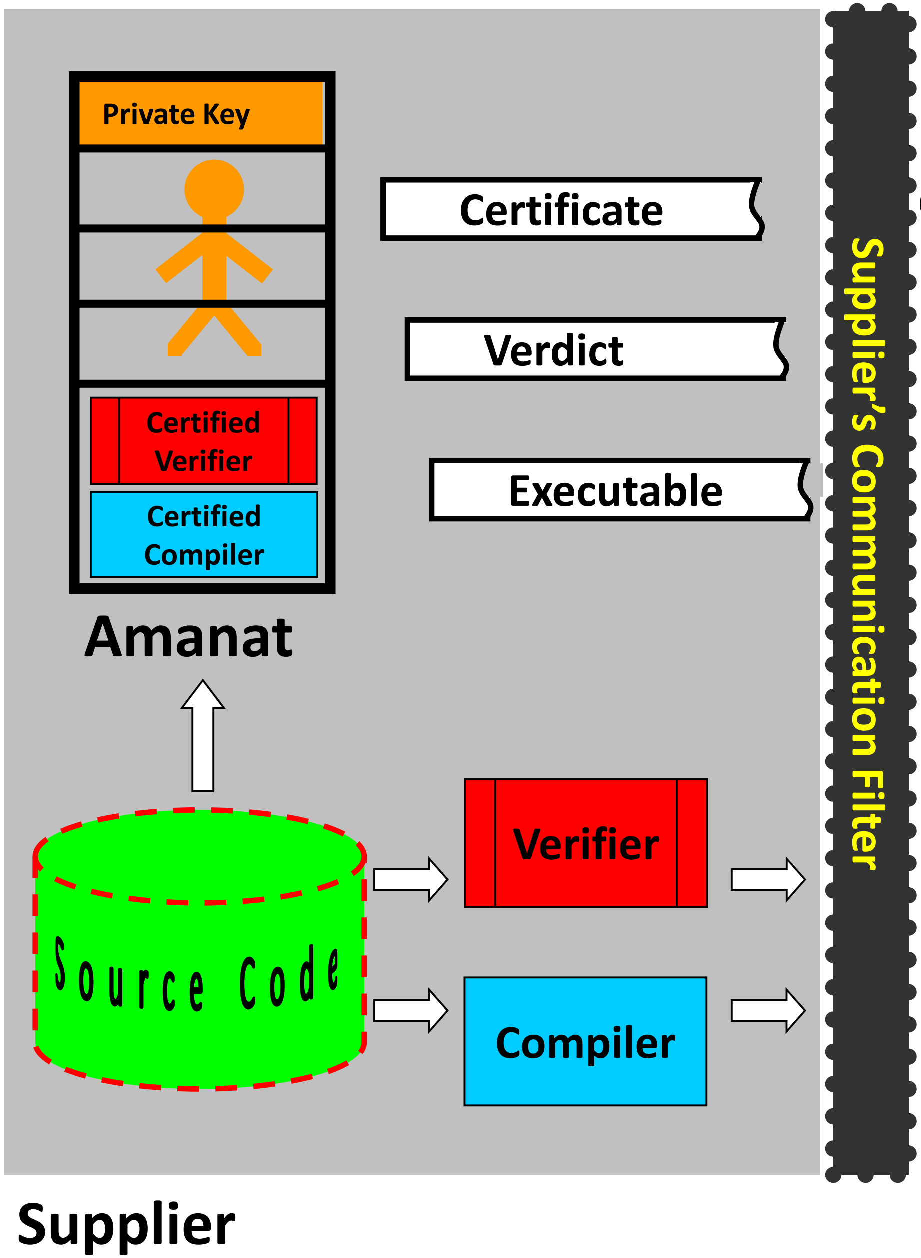


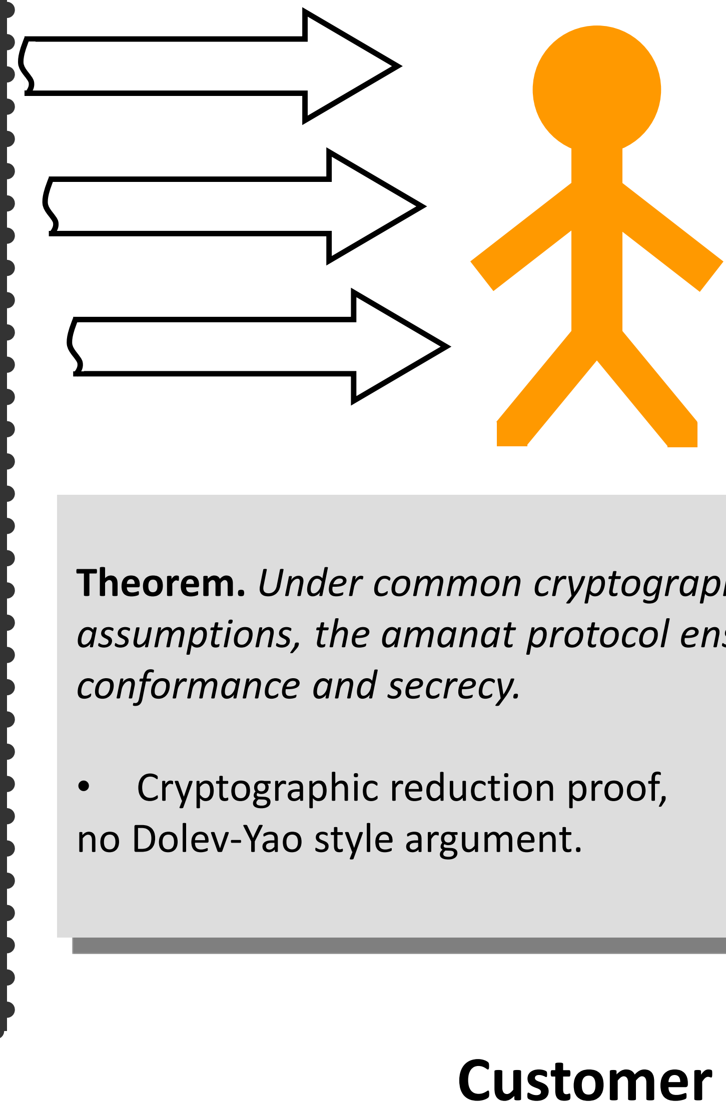
Public Key Cryptography

- *amanat* has private customer key

balance of power

- supplier can read and censor
- customer can validate origin





Theorem. *Under common cryptographic assumptions, the amanat protocol ensures conformance and secrecy.*

- Cryptographic reduction proof, no Dolev-Yao style argument.

Verification Tasks for the Amanat

Original motivation:

Apply **software model checkers** such as SLAM, BLAST, MAGIC

Method not confined to model checking

… not even to automated analysis tools

Unique condition

Truth of verification **verdict checkable by amanat**

e.g., amanat checks correctness of formalized manual proof

Verification Tasks for the Amanat

i. check a (semi)manual proof e.g. in ISABELLE, PVS, Coq, etc.

ii. apply static analysis tools – ASTREE, TVLA, …

iii. evaluate w…

iv. generate and execute white box test cases

v. validate tha… accompani… to coverage criteria

vi. check the code is syntactically safe, e.g. using lint

vii. compute numerical quality and quantity measures e.g. LOC, nesting depth etc.

viii. compare two versions of the source code and quantify the difference

ix. check presence of 3rd party IP, e.g. libraries

x. … algorithms

xi. validate that the source is well …

xii. …er has put …ier name on the source code

xiii. validate development steps by analyzing the CVS tree

xiv. ensure compatibility of source code to language standards

xv. …

Supplier bears the burden of proof.

can provide auxiliary information for amanat e.g. abstraction function, proof etc.

Physical Integrity of the Amanat

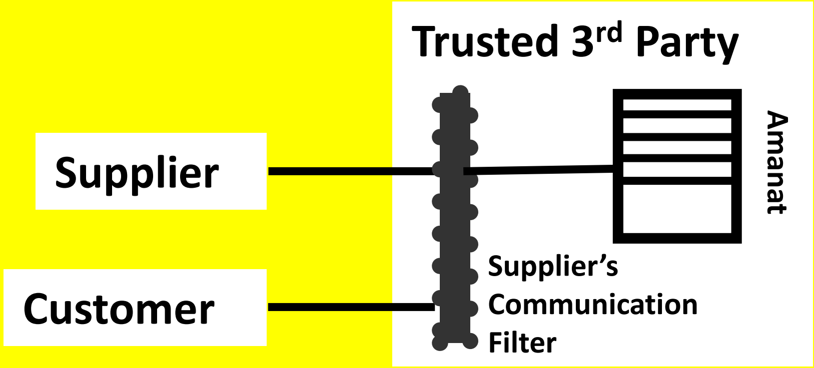
Amanat owns a *secret* – the private customer key

Amanat is a black box

→ Reverse engineering and physical monitoring prohibited

→ Need simple hardware solution

Scenario A



3rd party only ensures physical integrity

Amanat's communication hardwired through communication filter

Problem IP leaves supplier site

Physical Integrity **of the Amanat**

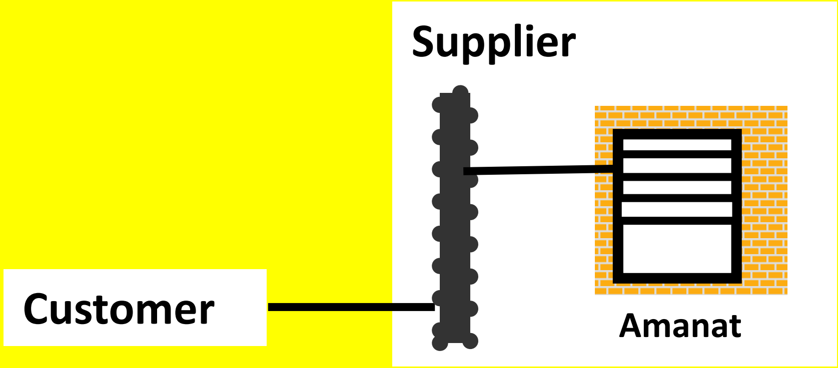
Amanat owns a *secret* – the private customer key

Amanat is a black box

→ Reverse engineering and physical monitoring prohibited

→ Need simple hardware solution

Scenario B



Amanat protected by physical seal

Regular checks by customer, 3rd party.
Alarm system, sealed hardware.

<u>No rational incentive to break seal.</u>

Verification Across Intellectual Property Boundaries

Part II:
Details of the Protocol

Tool Landscape

- Compiler: source → exec

source can be directory tree, compiler a make command

- Verifier: source → log_{Sup}, log_{Cus}

log_{Sup} is “internal” verdict
log_{Cus} is “external” verdict

Specifications are part of source, output into log_{Cus} together with verification verdict

All auxiliary information is part of source, provided by Sup
e.g. command line parameters, code annotations, abstraction functions etc.

Security Tool Landscape

Asymmetric encryption and signing scheme
(Cramer/Shoup 2000)

Key pair $< K_{pri}, K_{pub} >$

$c = K_{pub}(m)$ ………………………………... encryption of m by K_{pub}

$m = K_{pri}(K_{pub}(m))$

$s = csign(K_{pri}, m)$ ………………………... signature of m with key K_{pri}

$cverify(K_{pub}, m, s)$ ………………………... succeeds if s is valid signature

Security Tool Landscape

Asymmetric encryption and signing scheme (Cramer/Shoup 2000)

Cryptographic primitives for signing messages are ***randomized algorithms***

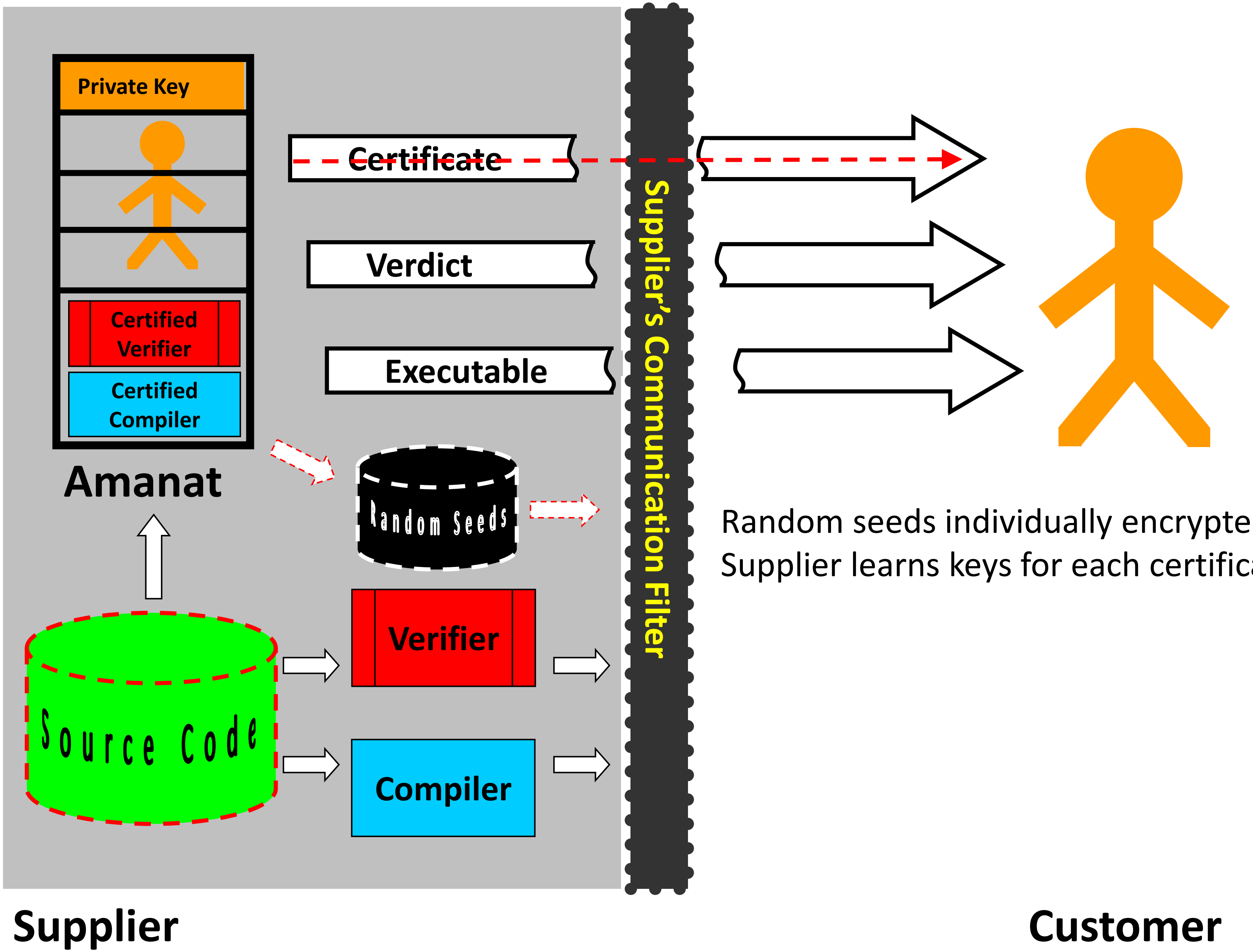
Countermeasure to algebraic attacks

Window to leak information from Ama to Cus:
Instead of random values, Ama can employ non-random bits describing the source code.

→ Protocol Design
*Ama **commits** random values before seeing the source code*

Similar to steganography.

Dolev-Yao style proofs not appropriate.



Random seeds individually encrypted →
Supplier learns keys for each certificate

Security Tool Landscape

Asymmetric encryption and signing scheme
(Cramer/Shoup 2000)

Key pair < K_{pri}, K_{pub}>

$c = K_{pub}(m)$ ……………………………... encryption of m by K_{pub}

$$m = K_{pri}(K_{pub}(m))$$

$s = csign(K_{pri},m,R)$ ……………..……... signature of m with key K_{pri}

$cverify(K_{pub}, m,s,R)$ …….…………...…. succeeds if s is valid signature

Amanat Protocol Overview

Installation Phase

Master key installation.

Session Initialization Phase

Tool Certification

Random Seed Generation

Certification Phase

Certification of Executables

Installation Phase

I1 ***Master Key Generation*** [Cus]

Cus generates the master keys $\langle K^m_{\mathsf{Cus}}, K^m_{\mathsf{Pub}} \rangle$ and initializes Ama with $\langle K^m_{\mathsf{Cus}}, K^m_{\mathsf{Pub}} \rangle$.

I2 ***Installation of the Amanat*** [Sup, Cus]

Ama is installed at Sup's site and Sup receives K^m_{Pub}.

Session Initialization Phase

S1 *Session Key Generation* [Cus, Sup]

Cus generates the session keys $\langle K_{\mathsf{Cus}}, K_{\mathsf{Pub}} \rangle$ and sends $K^m_{\mathsf{Pub}}(K_{\mathsf{Cus}})$ and K_{Pub} to Sup. Sup forwards $K^m_{\mathsf{Pub}}(K_{\mathsf{Cus}})$ and K_{Pub} unchanged to Ama.

S2 *Generation of the Tool Certificates* [Cus]

Cus computes the certificates

- $\mathsf{cert}_{\mathsf{Verifier}} = \mathsf{csign}(K_{\mathsf{Cus}}, \mathsf{Verifier})$ and
- $\mathsf{cert}_{\mathsf{Compiler}} = \mathsf{csign}(K_{\mathsf{Cus}}, \mathsf{Compiler})$.

Cus sends both certificates to Sup.

S3 *Supplier Validation of the Tool Certificates* [Sup]

Sup checks the contents of the certificates, i.e., Sup checks that

- $\mathsf{cverify}(K_{\mathsf{Pub}}, \mathsf{Verifier}, \mathsf{cert}_{\mathsf{Verifier}})$ and
- $\mathsf{cverify}(K_{\mathsf{Pub}}, \mathsf{Compiler}, \mathsf{cert}_{\mathsf{Compiler}})$ succeed.

If one of the checks fails, Sup aborts the protocol.

S4 *Amanat Tool Transmission* [Sup]

Sup sends to Ama both Verifier and Compiler as well as the certificates $\mathsf{cert}_{\mathsf{Verifier}}$ and $\mathsf{cert}_{\mathsf{Compiler}}$.

S5 *Amanat Validation of the Tool Certificates* [Ama]

Ama checks whether Verifier and Compiler are properly certified, i.e., it checks whether

- $\mathsf{cverify}(K_{\mathsf{Pub}}, \mathsf{Verifier}, \mathsf{cert}_{\mathsf{Verifier}})$ and
- $\mathsf{cverify}(K_{\mathsf{Pub}}, \mathsf{Compiler}, \mathsf{cert}_{\mathsf{Compiler}})$ succeed.

If this is not the case, then Ama refuses to process any further input.

S6 *Amanat Random Seed Generation* [Ama]

Ama generates

- a series of random seeds $R_1, \ldots, R_t$ together with a series of corresponding key pairs $\langle KR^1_{\mathsf{Cus}}, KR^1_{\mathsf{Pub}} \rangle, \ldots, \langle KR^t_{\mathsf{Cus}}, KR^t_{\mathsf{Pub}} \rangle$,
- encrypts the random seeds with the corresponding keys $KR^i_{\mathsf{Pub}}(R_i)$ for $i = 1, \ldots, t$, and
- initializes round counter $\mathsf{round} = 0$.

Ama then sends all $KR^i_{\mathsf{Pub}}(R_i)$ and KR^i_{Pub} for $i = 1, \ldots, t$ to Sup.

Certification Phase

C1 *Source Code Transmission* *[Sup]*

Sup sends source to Ama.

C2 *Source Code Verification by the Amanat* *[Ama]*

Ama computes

- the verdict $\langle \mathsf{log}_{\mathsf{Sup}}, \mathsf{log}_{\mathsf{Cus}} \rangle = \mathsf{Verifier}(\mathsf{source})$ of Verifier on source,
- the binary $\mathsf{exec} = \mathsf{Compiler}(\mathsf{source})$,
- increments the round counter round, and
- computes $\mathsf{cert} = \mathsf{csign}(K_{\mathsf{Cus}}, \langle \mathsf{exec}, \mathsf{log}_{\mathsf{Cus}} \rangle, \mathsf{R}_{\mathsf{round}})$.

Ama sends exec, $\mathsf{log}_{\mathsf{Sup}}$, $\mathsf{log}_{\mathsf{Cus}}$, cert, and $KR_{\mathsf{Cus}}^{\mathsf{round}}$ to Sup.

C3 *Secrecy Validation* [Sup]

Upon receiving exec, log_{Sup}, log_{Cus}, cert, and KR^{round}_{Cus}, Sup

- decrypts the random seed $R_{round} = KR^{round}_{Cus}(KR^{round}_{Pub}(R_{round}))$, and
- verifies that $\mathsf{cverify}(K_{Pub}, \langle \mathsf{exec}, \mathsf{log}_{Cus} \rangle, \mathsf{cert}, \mathsf{R}_{round})$ succeeds.

If the checks fails, Sup **concludes that the secrecy requirement was violated**, and refuses to further work with Ama.

Otherwise, Sup evaluates log_{Cus} and log_{Sup} and decides whether to deliver the binary exec, log_{Cus}, and cert to Cus in step **C4** or whether to abort the protocol.

C4 *Conformance Validation* [Cus]

Upon receiving exec, log_{Cus}, and cert, Cus verifies that $\mathsf{cverify}(K_{Pub}, \langle \mathsf{exec}, \mathsf{log}_{Cus} \rangle, \mathsf{cert})$ succeeds.

If the checks fails, Cus **concludes that the conformance requirement was violated**, and refuses to further work with Sup.

Otherwise Cus evaluates the contents of log_{Cus} and decides whether the verification verdict supports the purchase of the product exec.

Secrecy and Conformance

Theorem (Conformance) *If the amanat protocol terminates successfully, then exec and log_{Cus} must be produced from the same source in all but a negligible fraction of the protocol executions.*

Proof by reduction to cryptographic assumptions: violation of **semantic security** and **security against adaptive chosen message attacks.**

Theorem (Secrecy) *If the amanat protocol terminates successfully, then Cus cannot extract any piece of information from the source code which is not contained in exec and log_{Cus} .*

Proof by construction.

Cryptographic Assumptions

Semantic Security

K(c) has no more tractable information than |K(c)|.

All information which a probabilistic polynomial time algorithm can compute from K(c) can also be computed from |K(c)| in probabilistic polynomial time.

Security against Adaptive Chosen Message Attacks

Access to signing mechanism does not help circumvent signing.

Attacker has access to an oracle which signs arbitrary messages. Can the attacker in probabilistic polynomial time sign some new message without consulting the oracle?

Proof Outline: Secrecy

Theorem (Secrecy) *If the amanat protocol terminates successfully, then Cus cannot extract any piece of information from the source code which is not contained in exec and* log_{Cus} .

Proof by construction. We can show that every information passed to *Cus* can be computed without knowing source.

$$(\text{exec}, \text{Log}_{Cus}, \text{cert})$$

$$\text{cert} = \text{csign}(K_{Cus}, \langle \text{exec}, \text{log}_{Cus} \rangle, R_{round})$$

Proof Outline: Conformance

Theorem (Conformance) *If the amanat protocol terminates successfully, then exec and* log_{Cus} *must be produced from the same source in all but a negligible fraction of the protocol executions.*

Proof by reduction to cryptographic assumptions: violation of **semantic security** and **security against adaptive chosen message attacks.**

Assuming Conformance does not hold, we construct in 3 steps an adaptive chosen message attack against the signing scheme.

Verification Across Intellectual Property Boundaries

Part III: Conclusions

Verification Across Intellectual Property Boundaries

trusted verification w/o violation of IP rights

IP holder controls information flow

secrecy intuitive to management

simple cryptographic primitives

Future work: certification of COTS software ?

Thank you for your attention !

Questions?