THE PROTOCOL LIFECYCLE

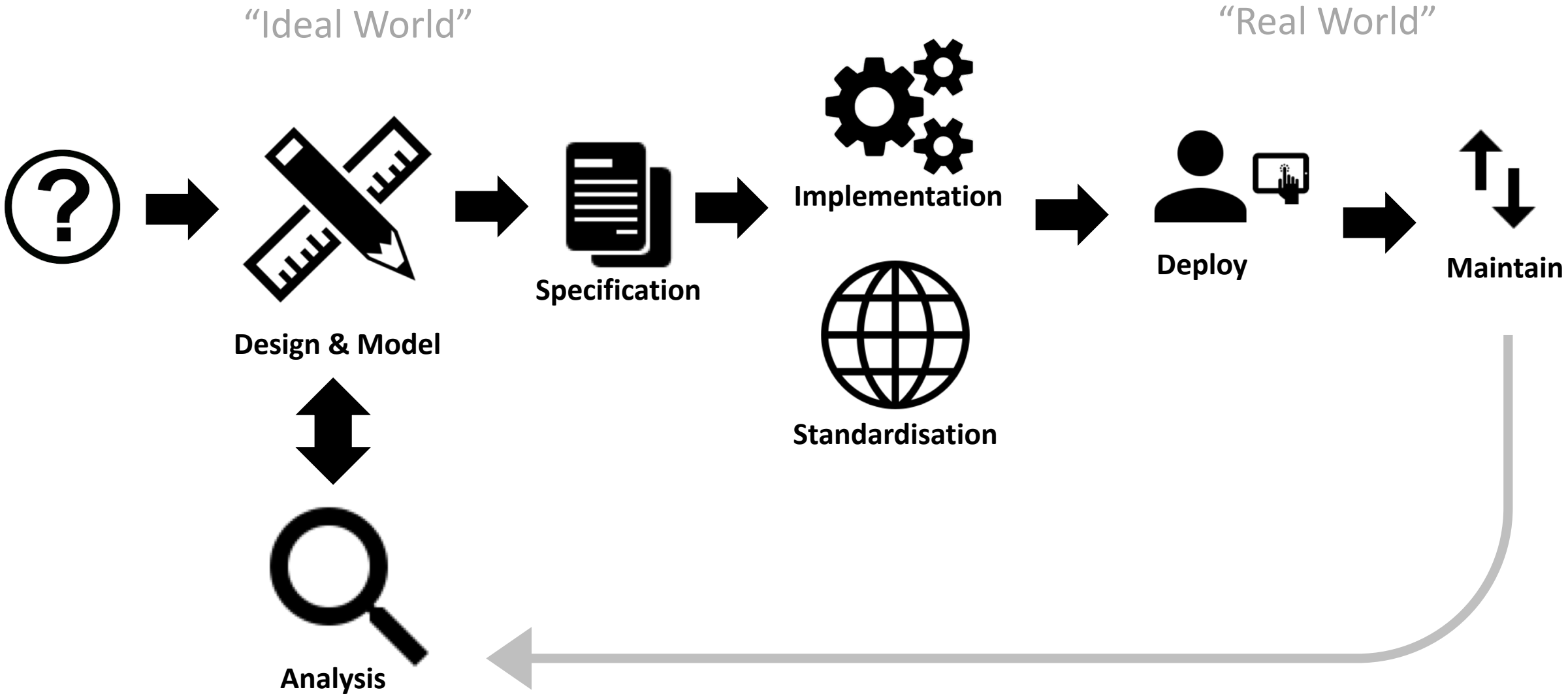
Roel Peeters

KU LEUVEN

My background

- Design of authentication protocols, systems
- Privacy (untraceability) in RFID
- DB protocols
- SW Implementation
- ePassport projects
- CTO of n-Auth: Mobile authentication university spin-off

Lifecycle


“Ideal World”
“Real World”
Design & Model
Specification
Implementation
Standardisation
Deploy
Maintain
Analysis

… in reality

Chaotic design phase

Convergence

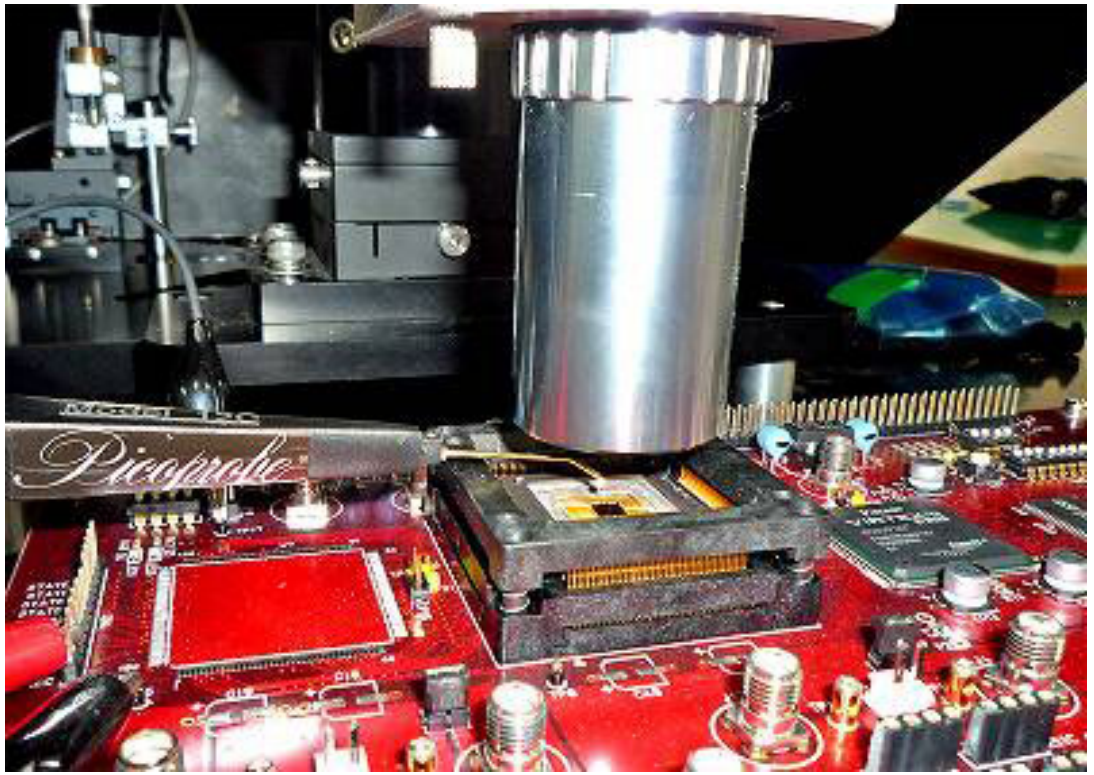
DESIGN & MODEL

Assumptions, security models, definitions

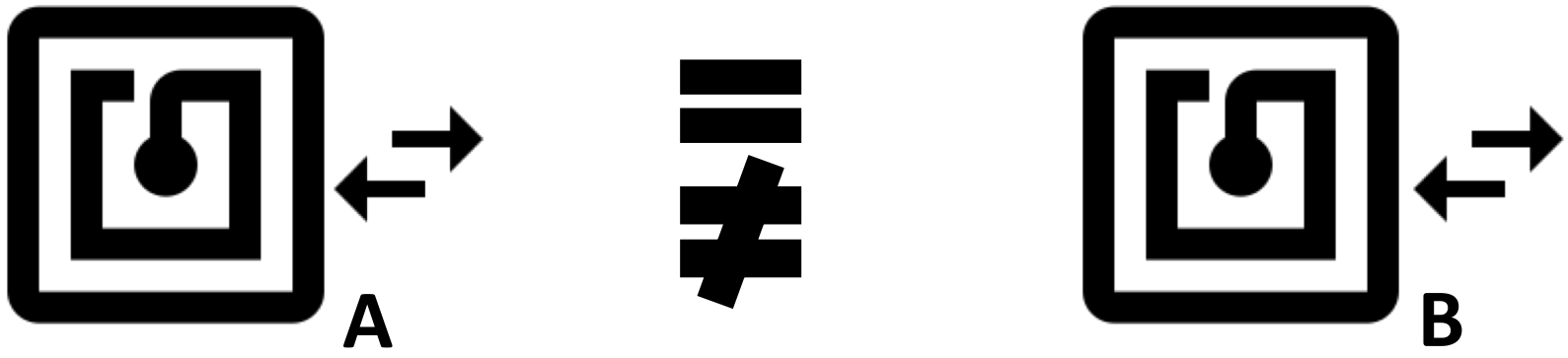
Invasive attacks

e.g. extracting keys, breaking HW, key leakage, side channels...

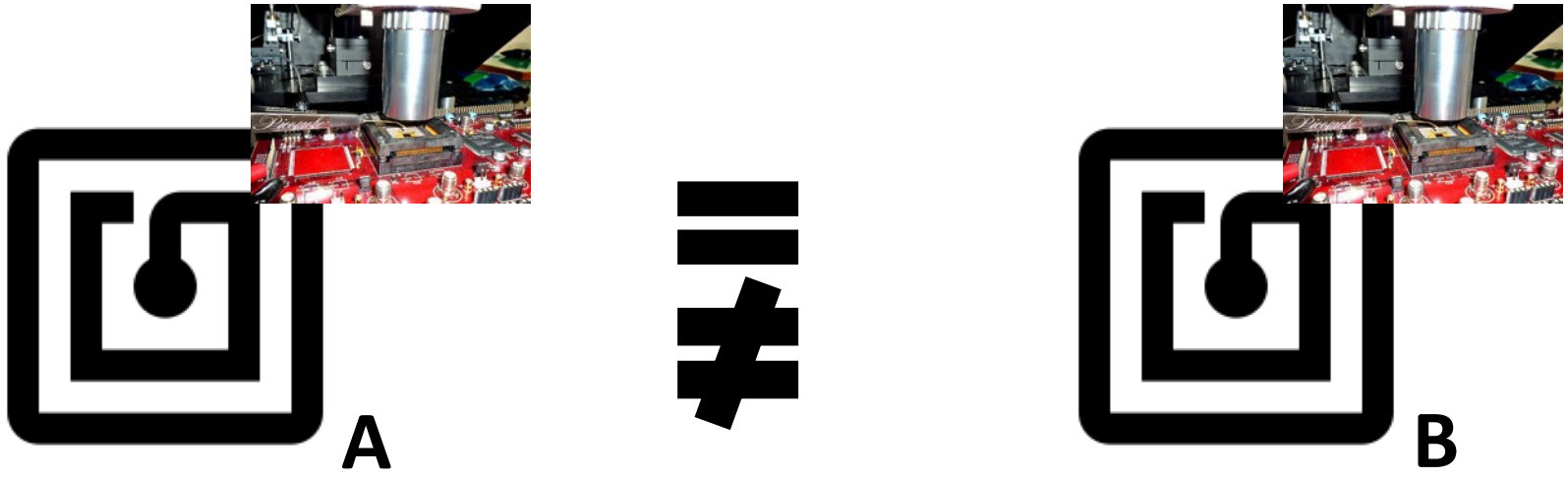
- Model or not?
- Don’t overdo it...



(Un)linkability



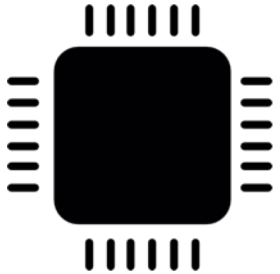
(Un)linkability



The Terrorist Fraud case

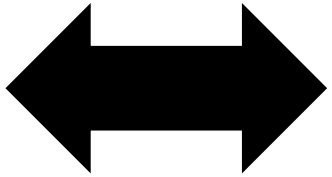
Related: cloning of prover (i.e. extracting keys)

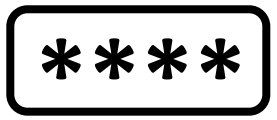
Protection:

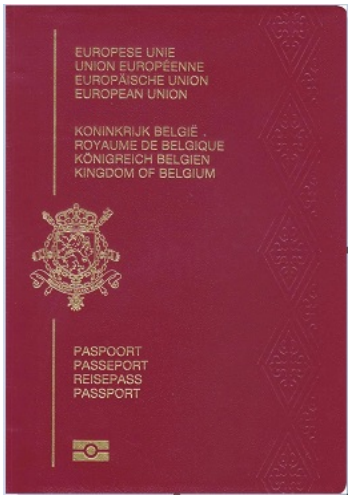


Typical TF-resistance: leak parts of key

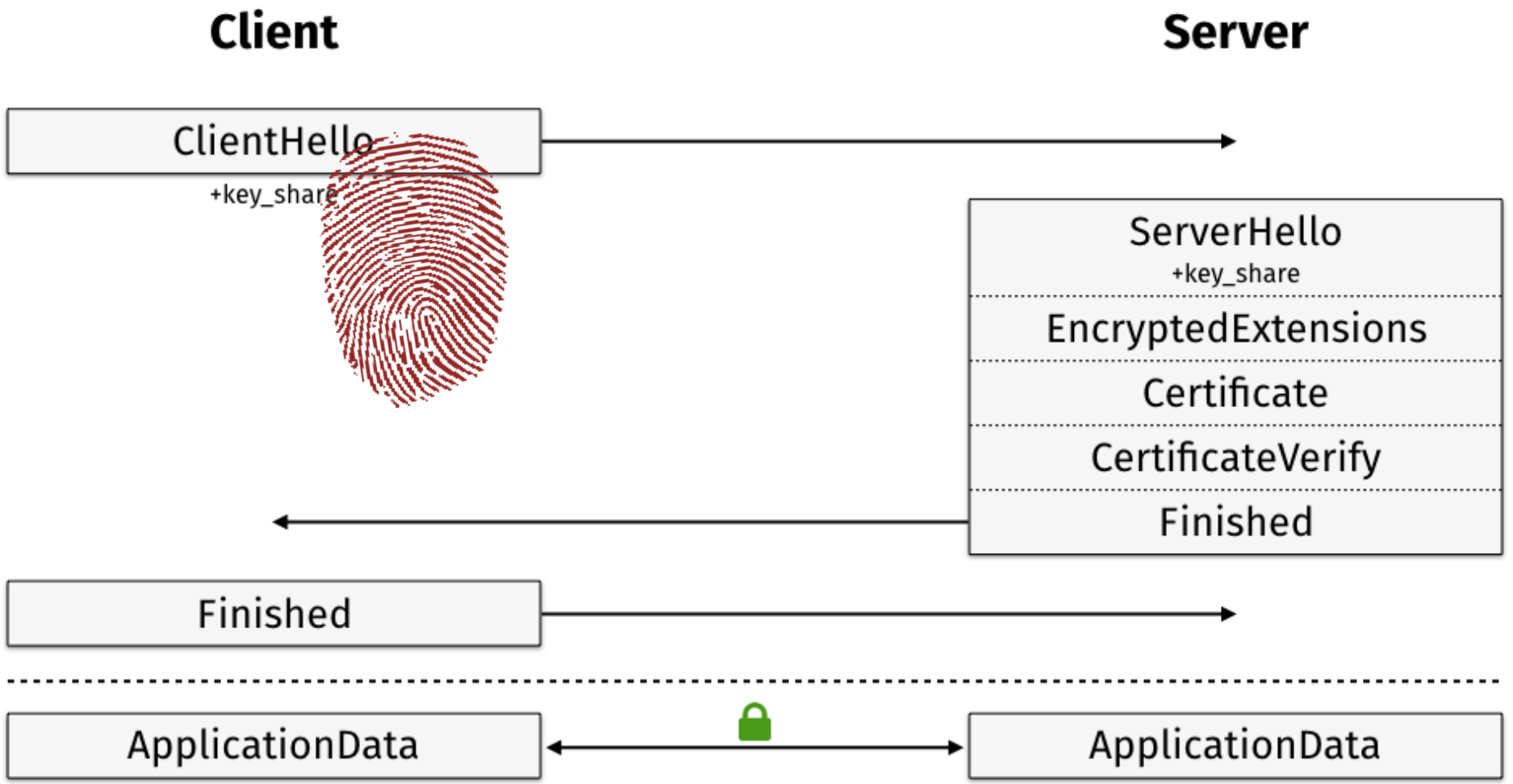
User vs device authentication





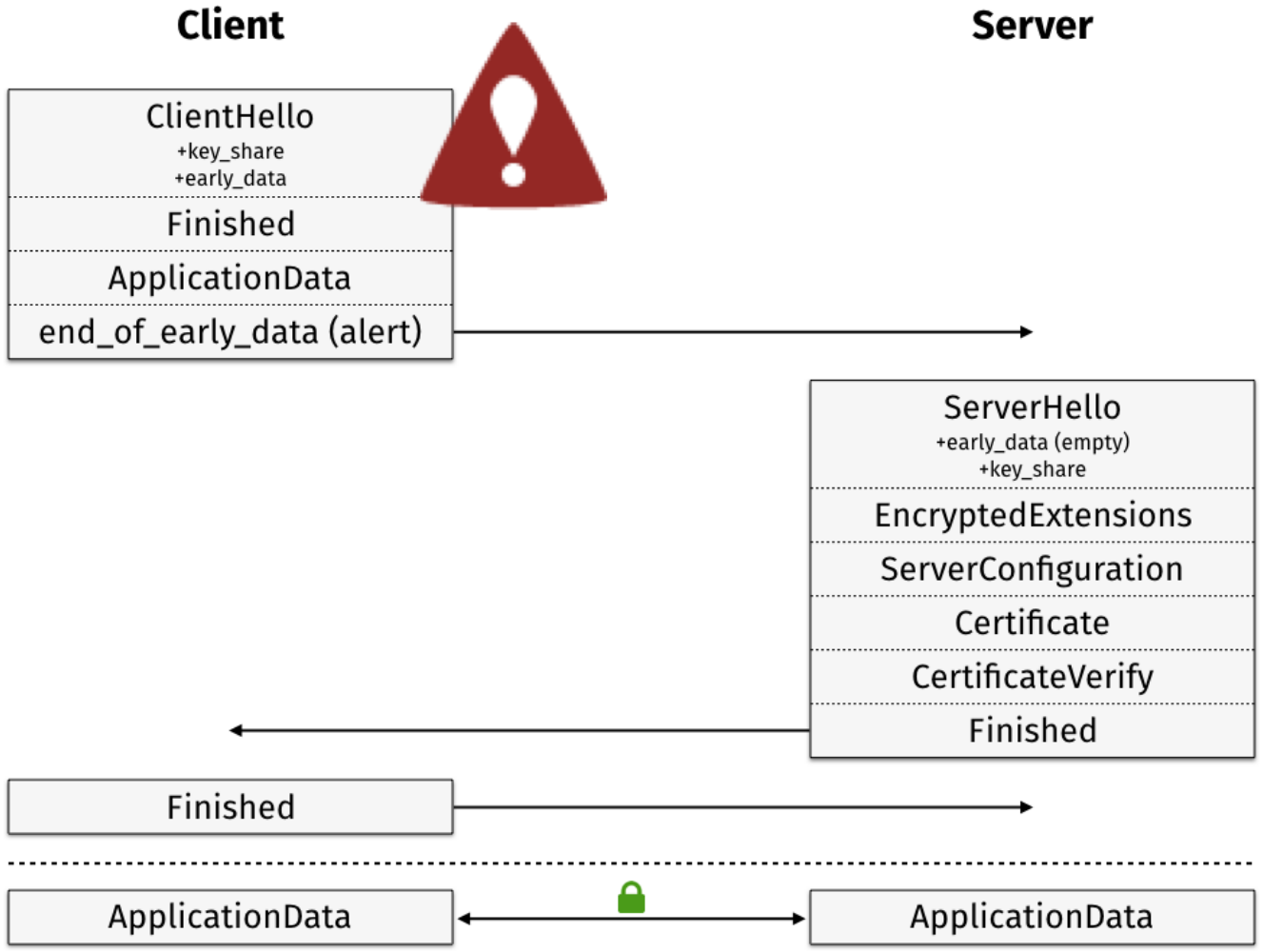


Tradeoffs – privacy vs efficiency

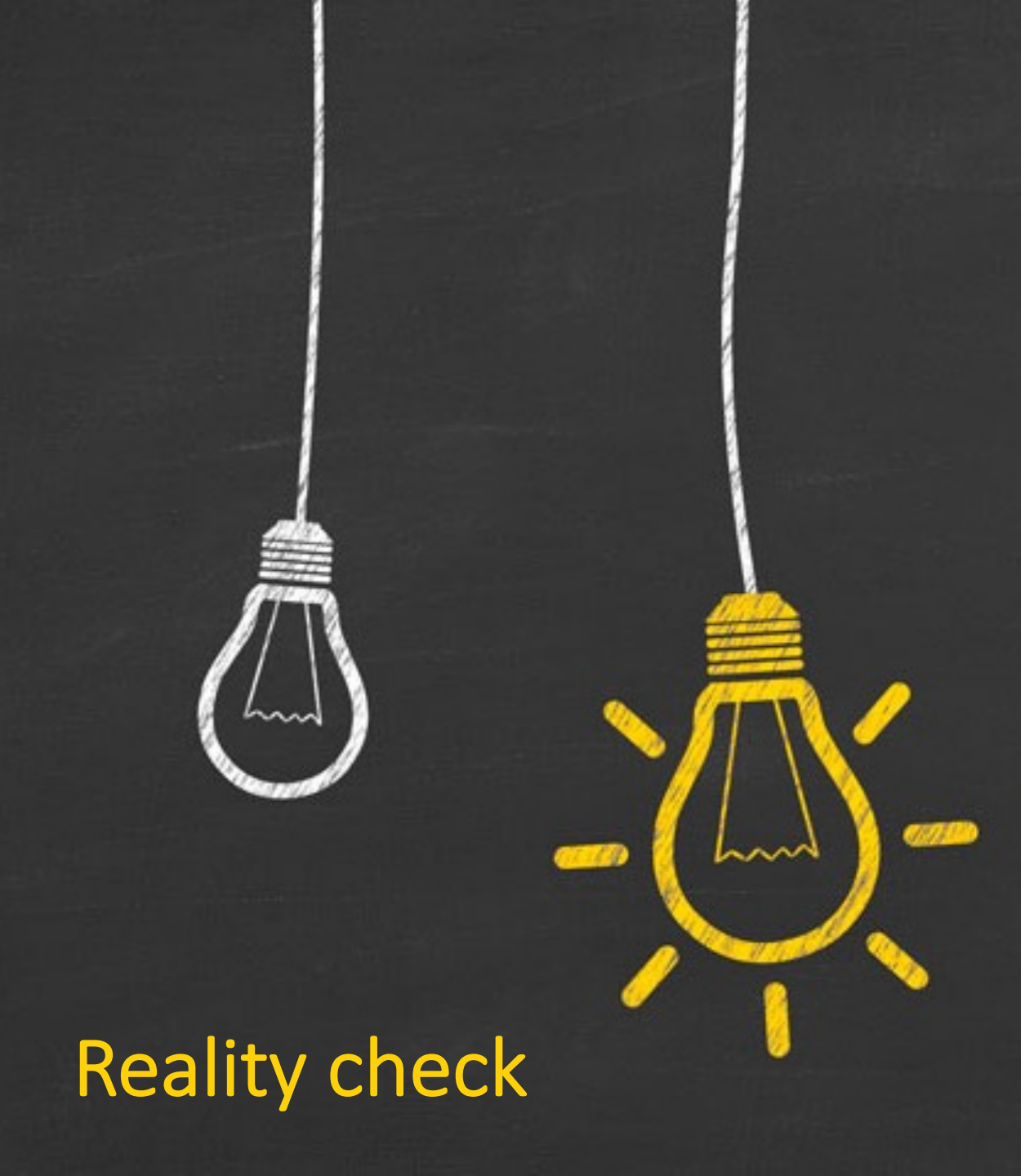


Src: timtaubert.de

Tradeoffs – security vs efficiency



Src: timtaubert.de



- Unrealistic properties/models
- Know model boundaries
- Look outside model
- Trade-offs

STANDARDISATION

SC27 / SC31 / SC37 / TC68 / ...


6 stages
Action
1
Proposal
NP
Proposal to start a new project
2
Preparatory
WD *
Expert consensus within working group
3
Committee
CD *
Committee consensus
4
Enquiry
DIS
National consensus
5
Approval
FDIS *
YES or NO vote
6
Publication
ISO
International Standard

ISO/IEC DIR 2

Edition 6.0 2011-04

ISO/IEC Directives Part 2

Rules for the structure and drafting of International Standards

Internet Engineering Task Force (IETF)
Request for Comments: 7282
Category: Informational
ISSN: 2070-1721

P. Resnick
Qualcomm Technologies, Inc.

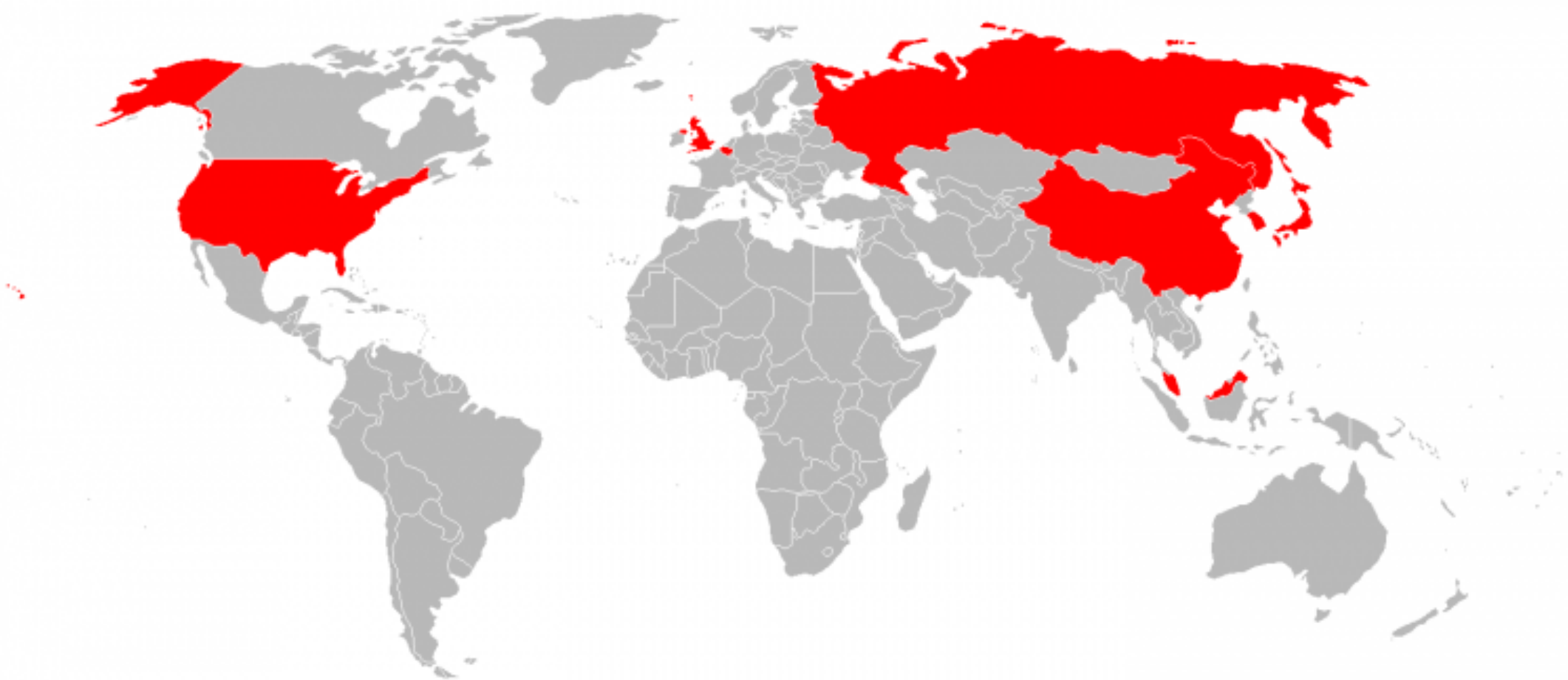
June 2014

On Consensus and Humming in the IETF

Abstract

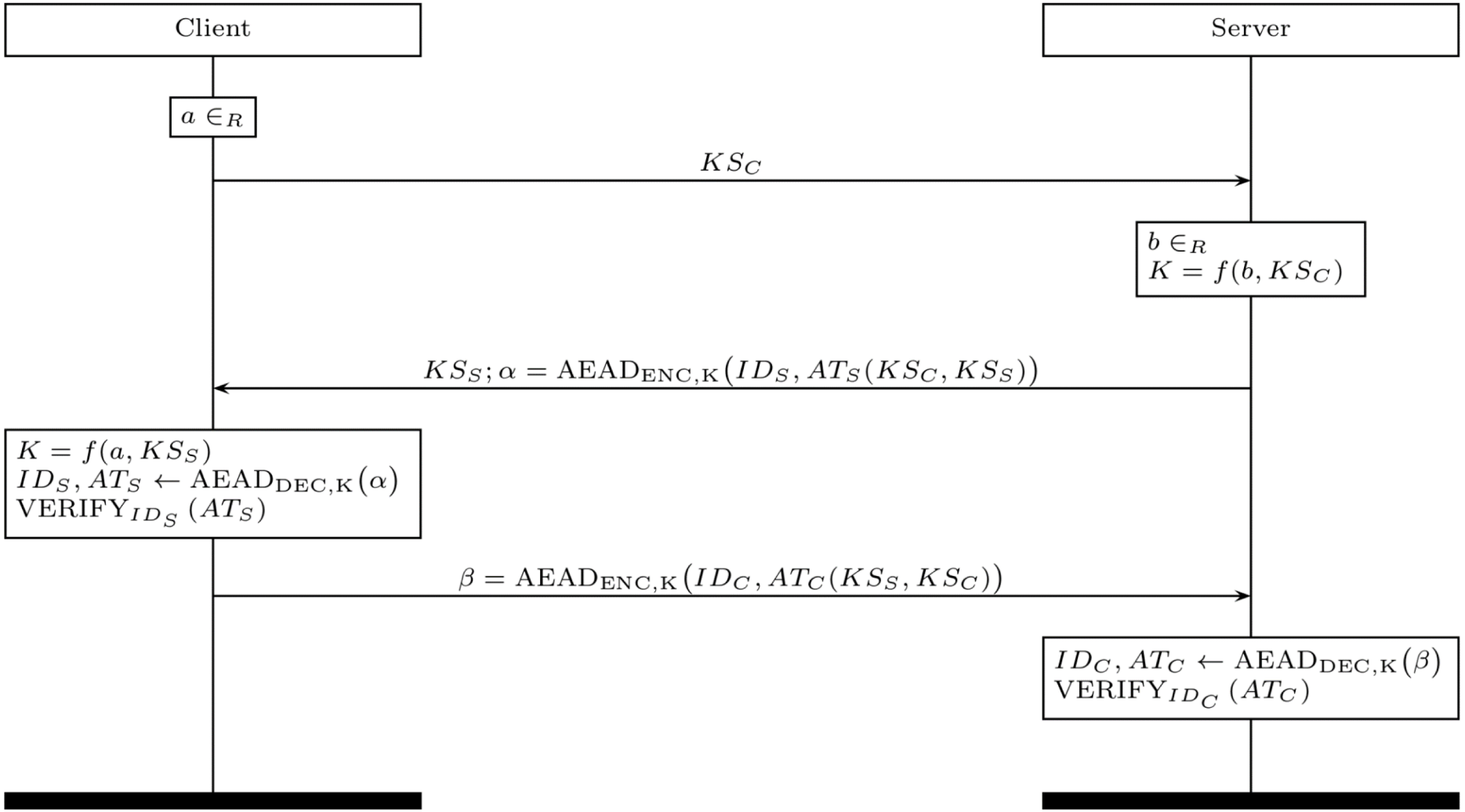
The IETF has had a long tradition of doing its technical work through a consensus process, taking into account the different views among IETF participants and coming to (at least rough) consensus on technical matters. In particular, the IETF is supposed not to be run by a "majority rule" philosophy. This is why we engage in rituals like "humming" instead of voting. However, more and more of our actions are now indistinguishable from voting, and quite often we are letting the majority win the day without consideration of minority concerns. This document explains some features of rough consensus, what is not rough consensus, how we have gotten away from it, how we might think about it differently, and the things we can do in order to really achieve rough consensus.

Politics

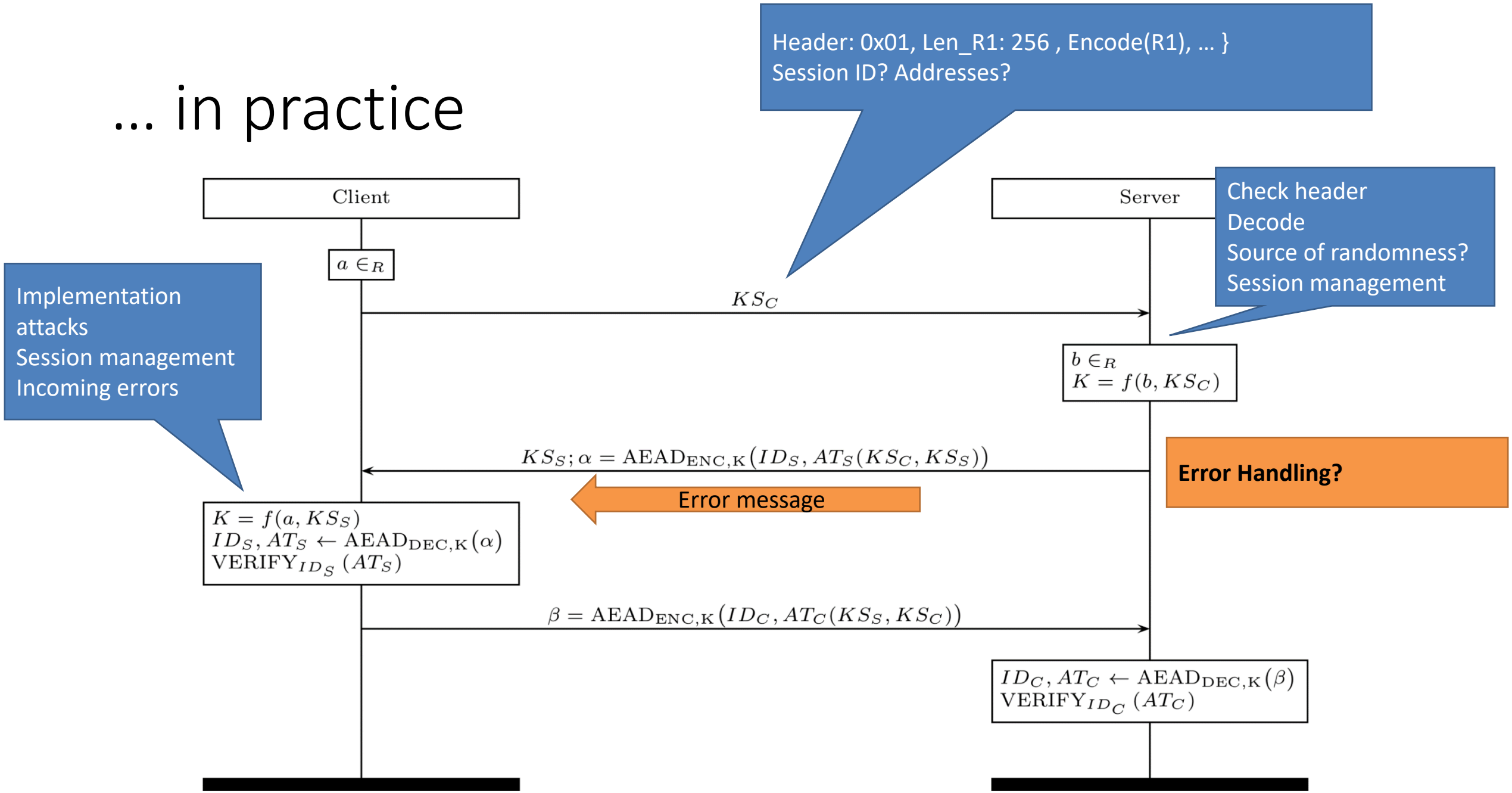


SPECIFICATION

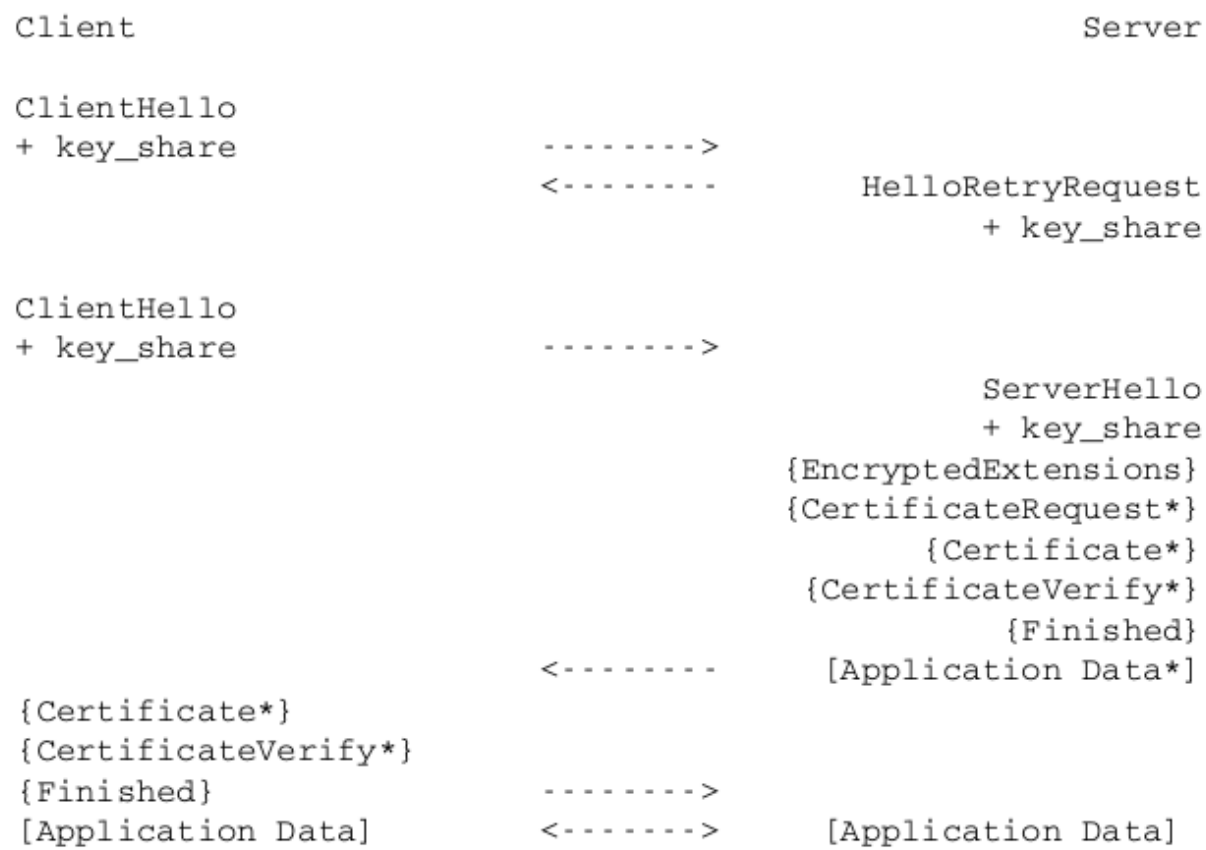
Protocols… in theory



… in practice


Header: 0x01, Len_R1: 256 , Encode(R1), … }
Session ID? Addresses?
Client
Server
Check header
Decode
Source of randomness?
Session management
Implementation attacks
Session management
Incoming errors
$a \in_R$
KS_C
$b \in_R$
$K = f(b, KS_C)$
$KS_S; \alpha = \text{AEAD}_{\text{ENC,K}}(ID_S, AT_S(KS_C, KS_S))$
Error Handling?
Error message
$K = f(a, KS_S)$
$ID_S, AT_S \leftarrow \text{AEAD}_{\text{DEC,K}}(\alpha)$
$\text{VERIFY}_{ID_S}(AT_S)$
$\beta = \text{AEAD}_{\text{ENC,K}}(ID_C, AT_C(KS_S, KS_C))$
$ID_C, AT_C \leftarrow \text{AEAD}_{\text{DEC,K}}(\beta)$
$\text{VERIFY}_{ID_C}(AT_C)$

A look…


Client
Server
ClientHello
+ key_share
HelloRetryRequest
+ key_share
ClientHello
+ key_share
ServerHello
+ key_share
{EncryptedExtensions}
{CertificateRequest*}
{Certificate*}
{CertificateVerify*}
{Finished}
[Application Data*]
{Certificate*}
{CertificateVerify*}
{Finished}
[Application Data]
[Application Data]

A look…

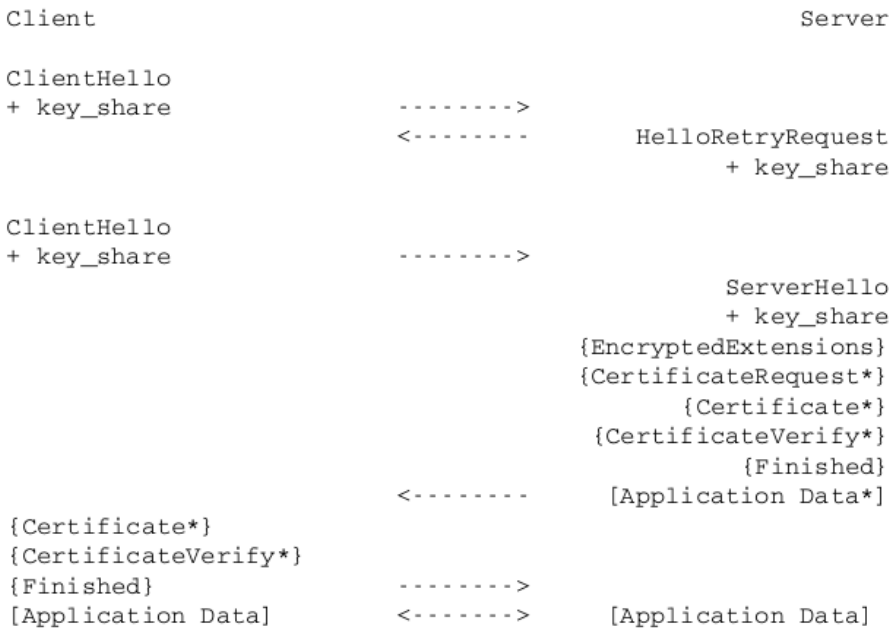
```
uint16 ProtocolVersion;
opaque Random[32];

uint8 CipherSuite[2];     /* Cryptographic suite selector */

struct {
    ProtocolVersion legacy_version = 0x0303;     /* TLS v1.2 */
    Random random;
    opaque legacy_session_id<0..32>;
    CipherSuite cipher_suites<2..2^16-2>;
    opaque legacy_compression_methods<1..2^8-1>;
    Extension extensions<0..2^16-1>;
} ClientHello;
```

A look…

The version and extensions fields have the same meanings as their corresponding values in the ServerHello. The server SHOULD send only the extensions necessary for the client to generate a correct ClientHello pair. As with ServerHello, a HelloRetryRequest MUST NOT contain any extensions that were not first offered by the client in its ClientHello, with the exception of optionally the "cookie" (see Section 4.2.2) extension.

Upon receipt of a HelloRetryRequest, the client MUST verify that the extensions block is not empty and otherwise MUST abort the handshake with a "decode_error" alert. Clients MUST abort the handshake with an "illegal_parameter" alert if the HelloRetryRequest would not result in any change in the ClientHello. If a client receives a second HelloRetryRequest in the same connection (i.e., where the ClientHello was itself in response to a HelloRetryRequest), it MUST abort the handshake with an "unexpected_message" alert.

A look…



Formal verification claimed!?

The version and extensions fields have the same meanings as their corresponding values in the ServerHello. The server SHOULD send only the extensions necessary for the client to generate a correct ClientHello pair. As with ServerHello, a HelloRetryRequest MUST NOT contain any extensions that were not first offered by the client in its ClientHello, with the exception of optionally the "cookie" (see Section 4.2.2) extension.

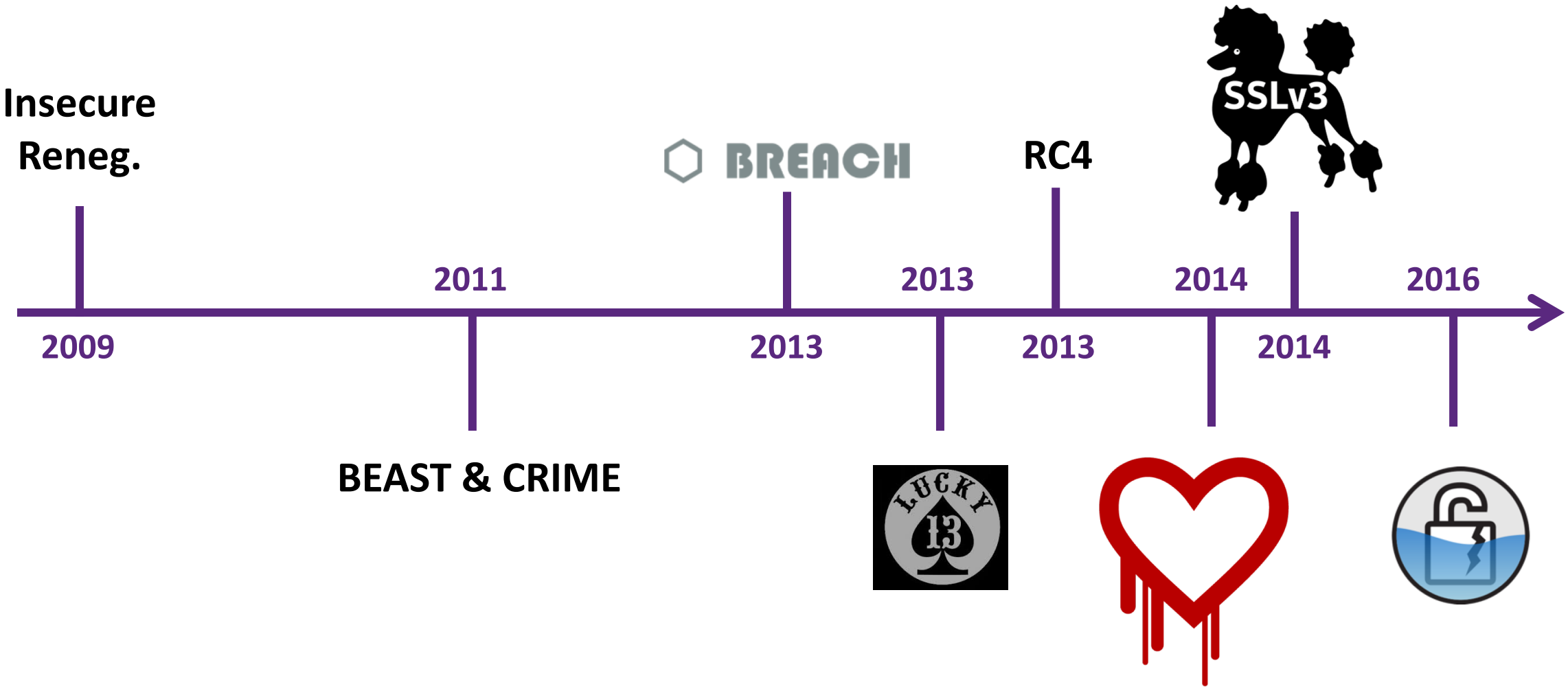
Upon receipt of a HelloRetryRequest, the client MUST verify that the extensions block is not empty and otherwise MUST abort the handshake with a "decode_error" alert. Clients MUST abort the handshake with an "illegal_parameter" alert if the HelloRetryRequest would not result in any change in the ClientHello. If a client receives a second HelloRetryRequest in the same connection (i.e., where the ClientHello was itself in response to a HelloRetryRequest), it MUST abort the handshake with an "unexpected_message" alert.

Specifications

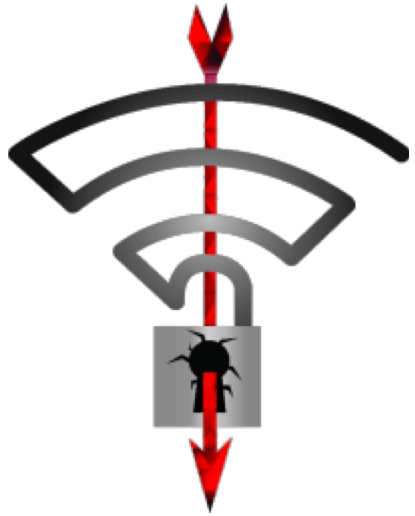
- Balance formalism with abstraction
- Most exact specification = formal spec
- Always capture security

IMPLEMENTATION

Typical problems



Typical problems



KRACK (2017)

Implementer

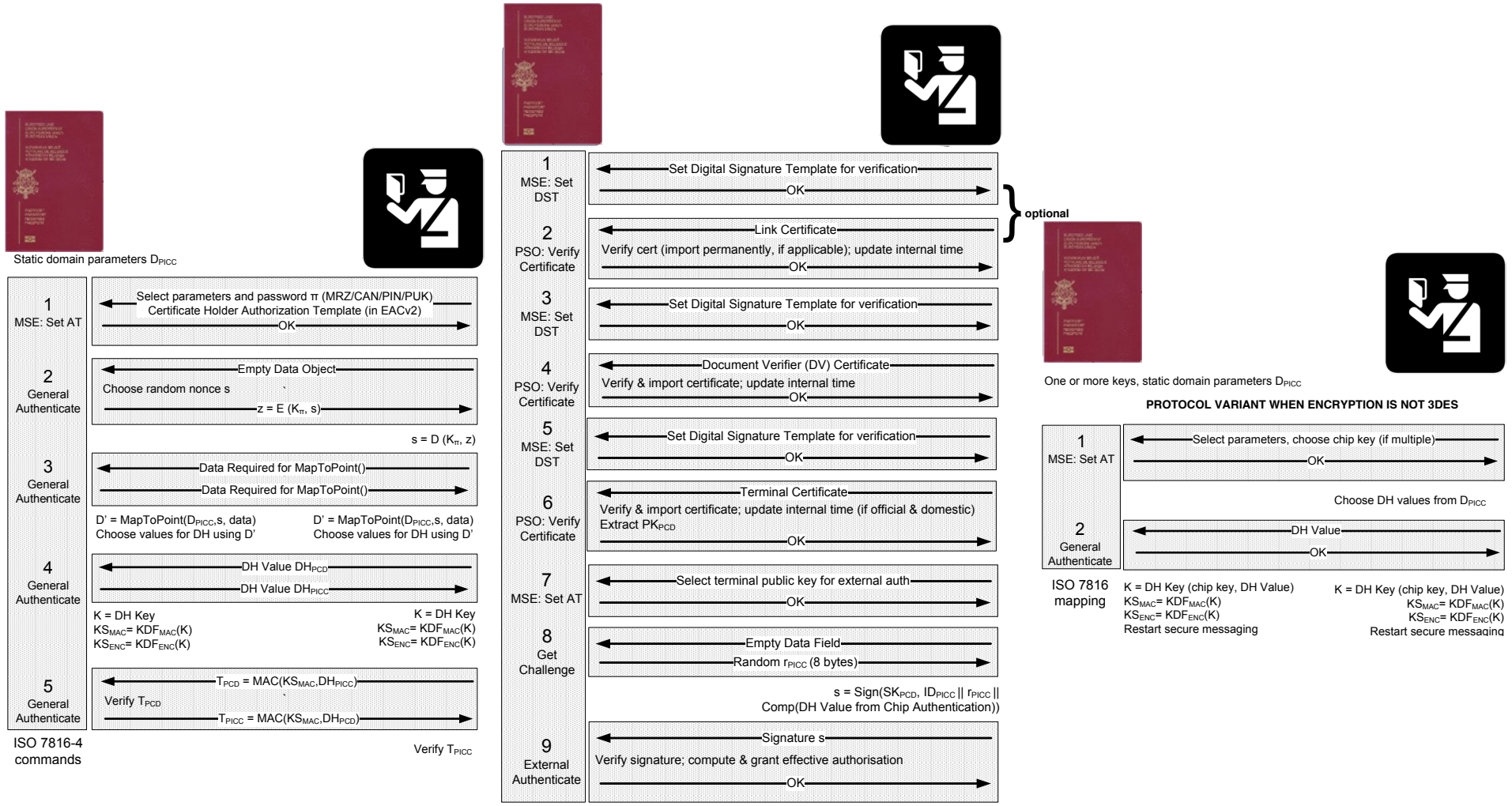
Does not know:

- Model, assumptions, crypto...
- ‘Trivial’ mistakes that have been made 100 times before
- What you really mean
- What the protocol is supposed to do
- Security properties

How to confuse an implementer

1. MUST This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
2. MUST NOT This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
3. SHOULD This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
4. SHOULD NOT ...
5. MAY This word, or the adjective "OPTIONAL", mean that an item is truly optional. ...

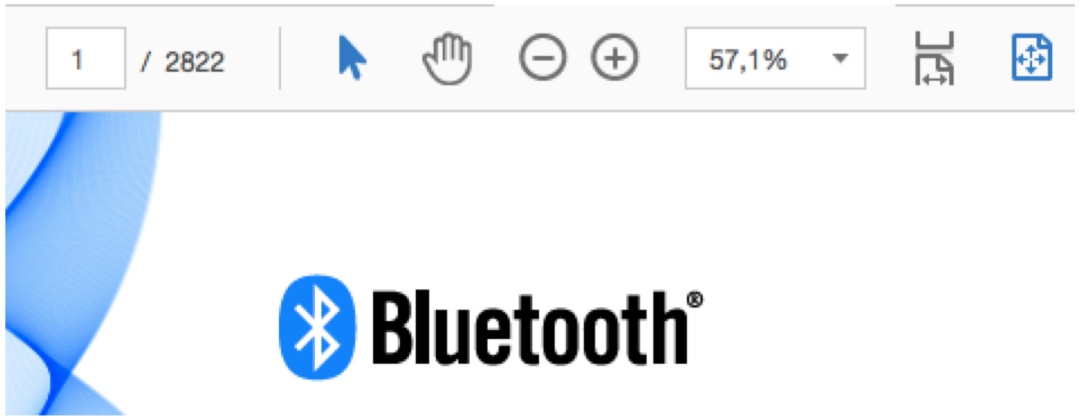
How to confuse an implementer



How to confuse an implementer

It can always be worse:

- EMV spec (biggest PKI in the world)
- Bluetooth spec

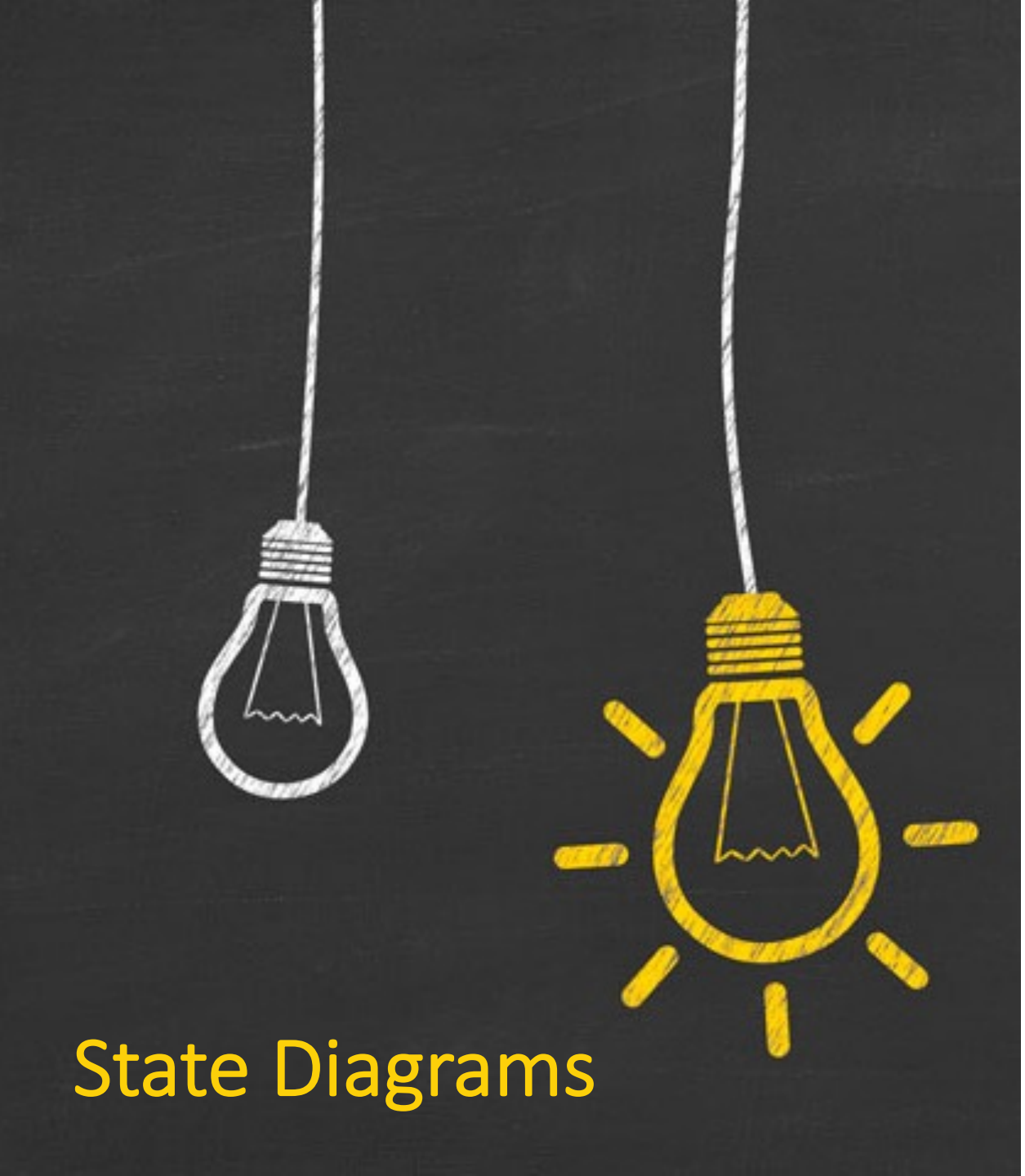


Field Name	Length	Description	Format
Recovered Data Header	1	Hex Value '6A'	b
Certificate Format	1	Hex Value '02'	b
Issuer Identifier	4	Leftmost 3-8 digits from the PAN (padded to the right with Hex 'F's)	cn 8
Certificate Expiration Date	2	MMYY after which this certificate is invalid	n 4
Certificate Serial Number	3	Binary number unique to this certificate assigned by the certification authority	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [5]	b
Issuer Public Key Algorithm Indicator	1	Identifies the digital signature algorithm to be used with the Issuer Public Key [5]	b
Issuer Public Key Length	1	Identifies the length of the Issuer Public Key Modulus in bytes	b
Issuer Public Key Exponent Length	1	Identifies the length of the Issuer Public Key Exponent in bytes	b
Issuer Public Key or Leftmost Digits of the Issuer Public Key	N_{CA} –36	If $N_I \leq N_{CA} - 36$, consists of the full Issuer Public Key padded to the right with $N_{CA} - 36 - N_I$ bytes of value 'BB' If $N_I > N_{CA} - 36$, consists of the $N_{CA} - 36$ most significant bytes of the Issuer Public Key [6]	b
Hash Result	20	Hash of the Issuer Public Key and its related information	b
Recovered Data Trailer	1	Hex value 'BC'	b

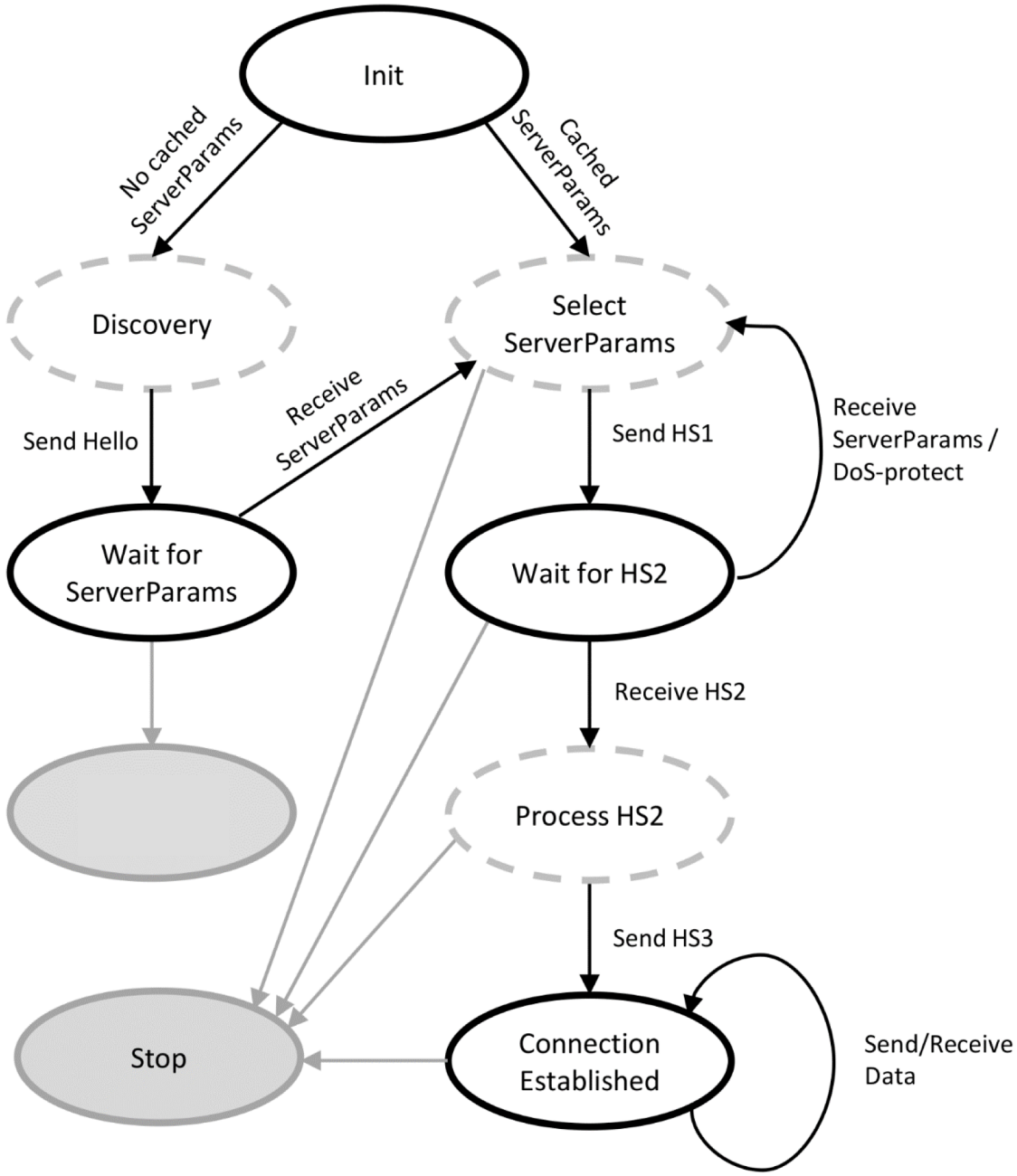
Table 6: Format of Data Recovered from Issuer Public Key Certificate

Goals

- Simple to implement
- Limit interpretation
- No exceptions
- Guidance on sec. properties



State Diagrams



Formal verification

.... of implementations?

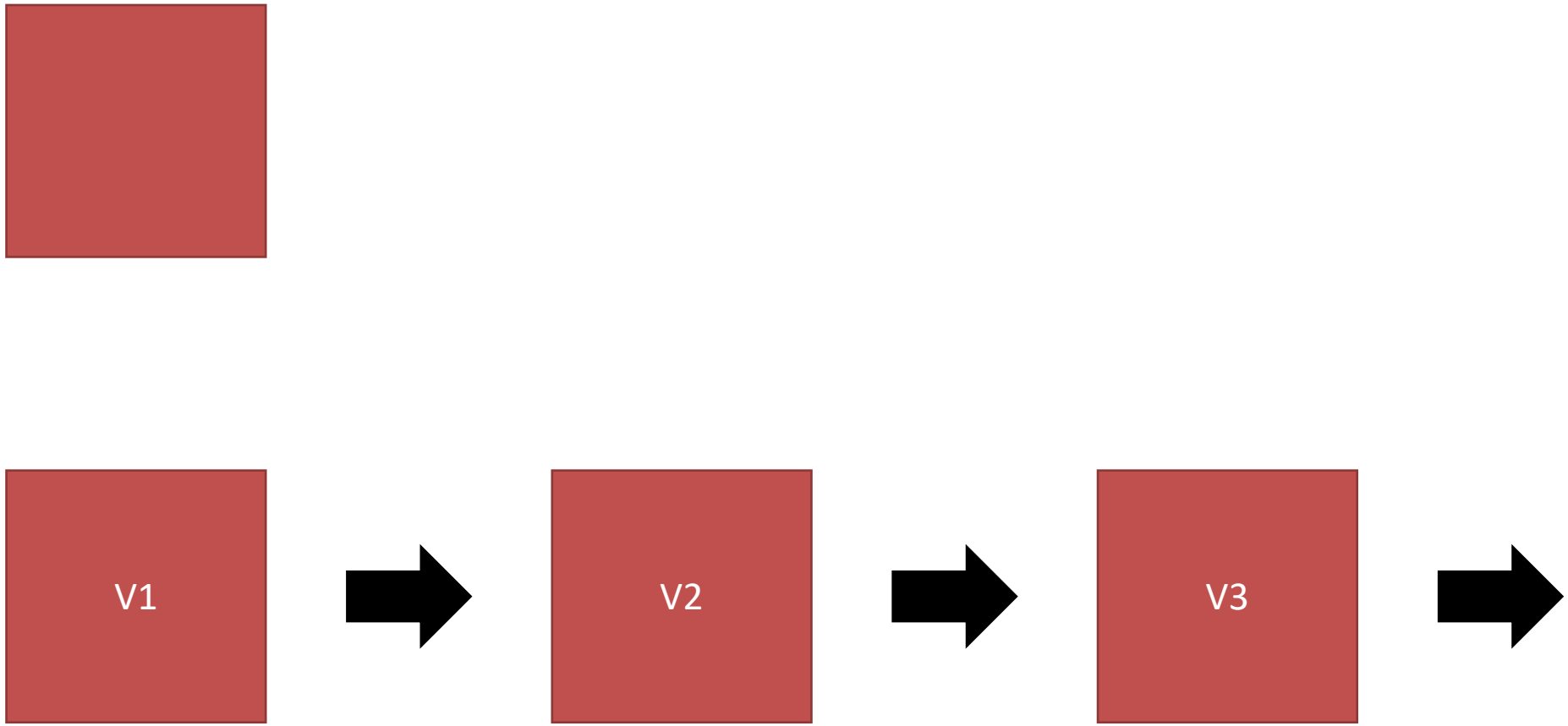
code

~

reference

MAINTENANCE

Versioning



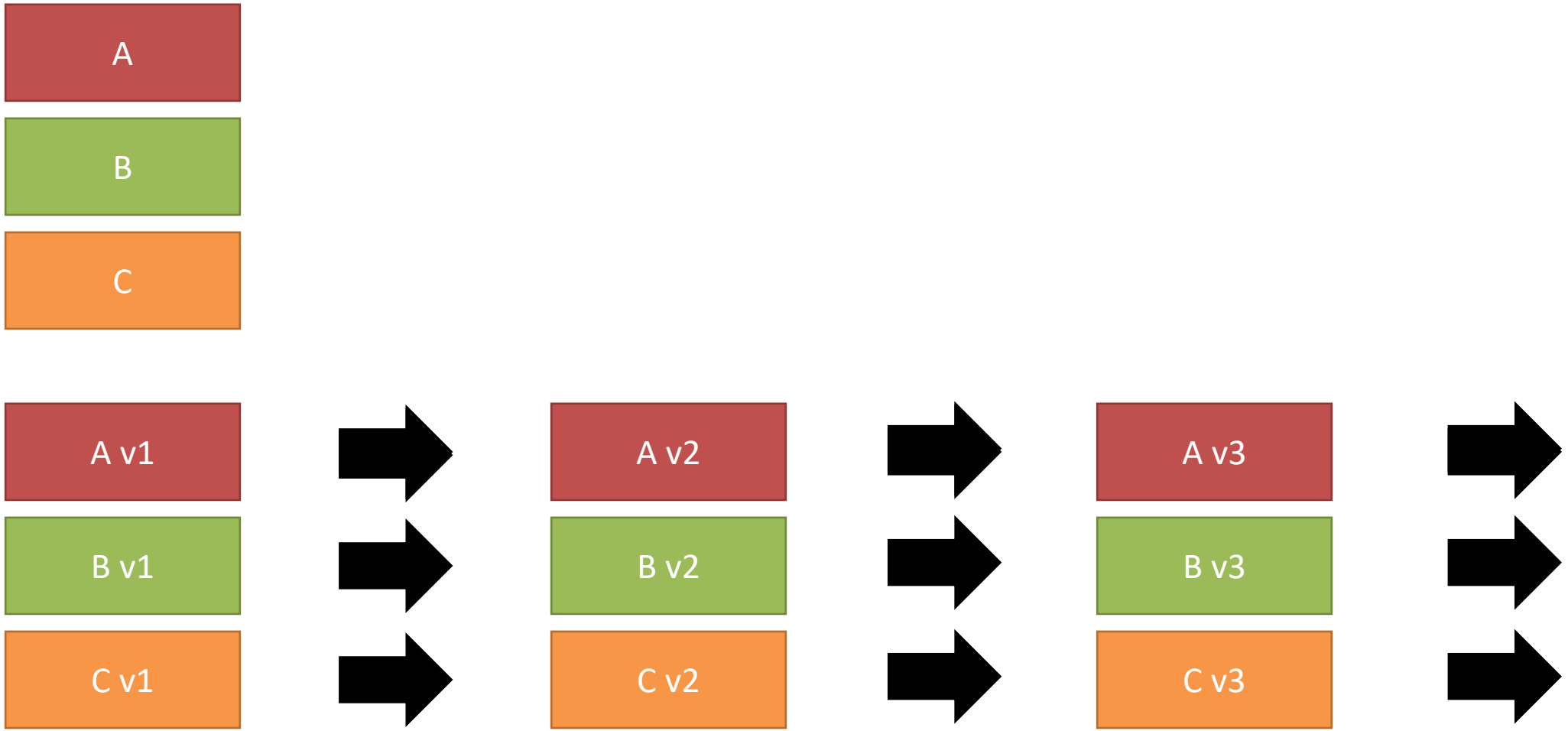
Don’t!

Future proof:

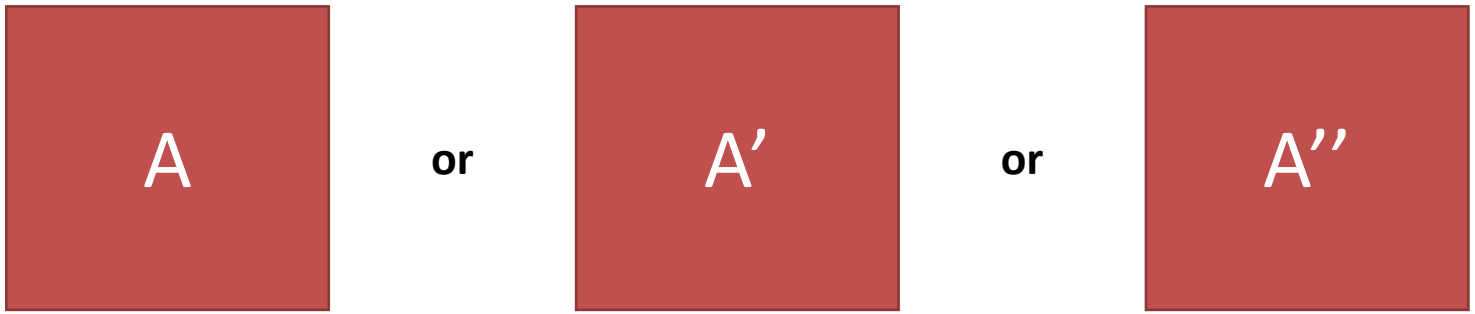
Authenticated version list

(as part of protocol or elsewhere)

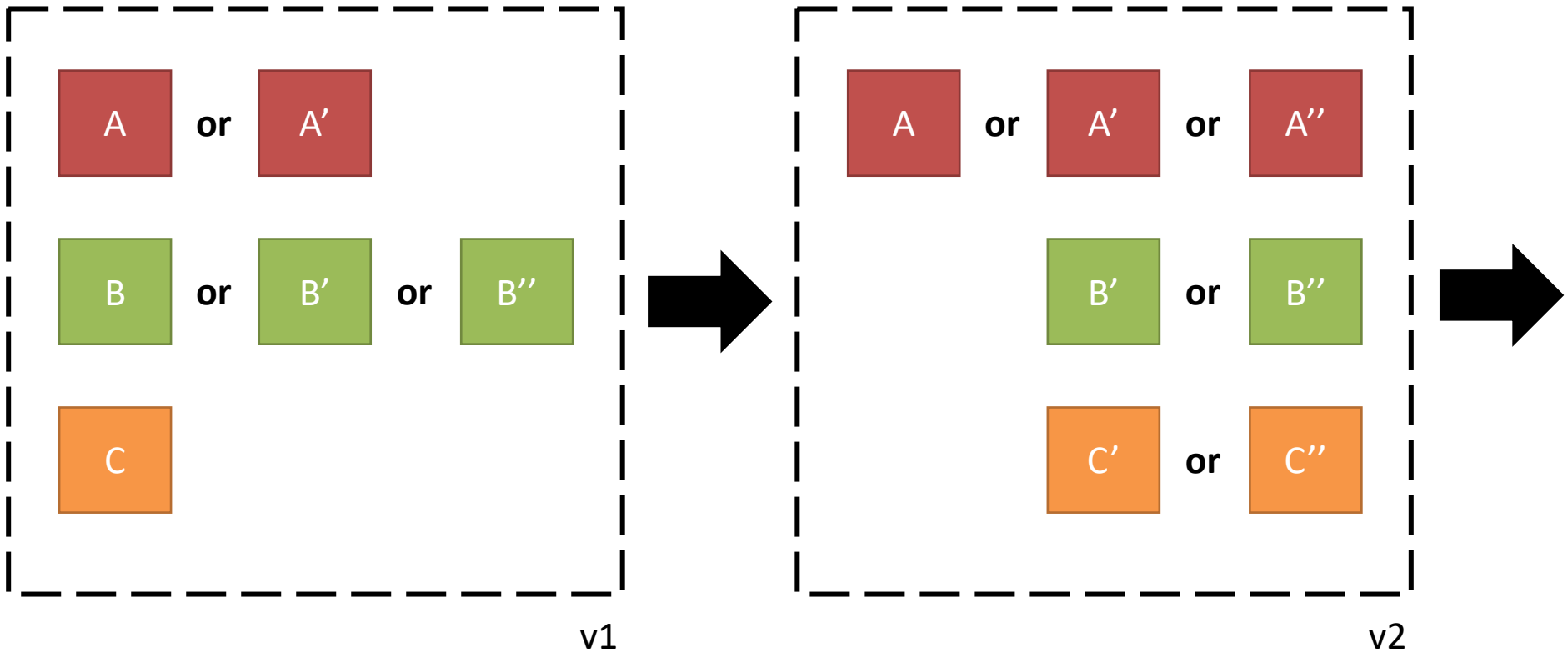
Modularity



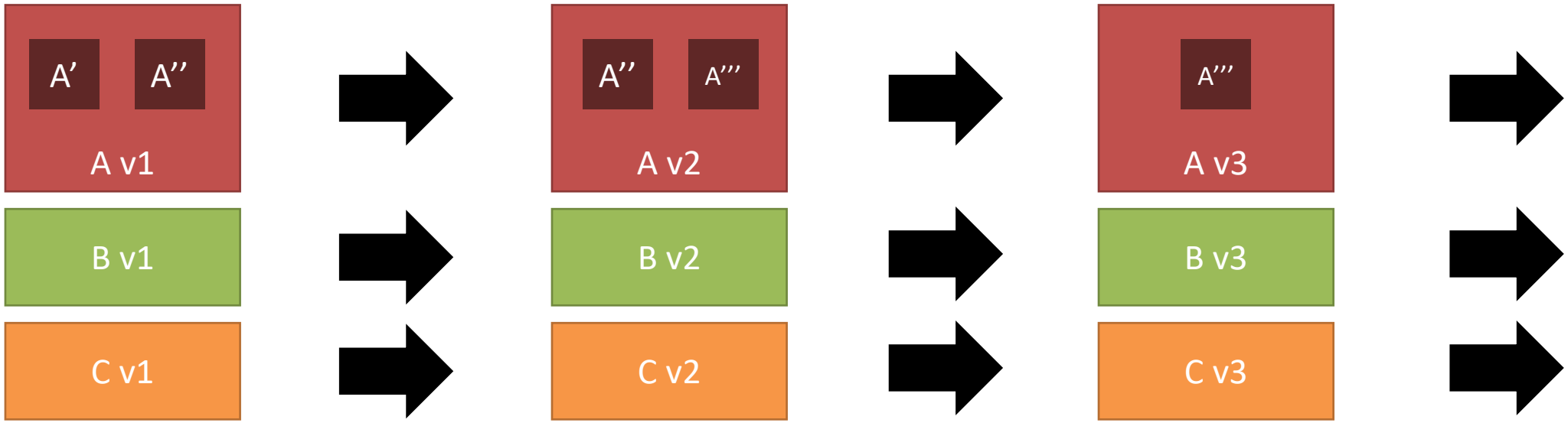
Choices



Zoo approach



Hybrid



Balance:

- Algorithm/protocol agility
- Robustness
- Simplicity

(always ensure security)