CAN DICTATORS BE GOOD?

(ONLY BENEVOLENT ONES, OF COURSE)

TALK FOR THE OCCASION OF GUIDO VAN ROSSUM'S DIJKSTA FELLOWSHIP

SAPE MULLENDER
CHIEF COOK AND BOTTLE WASHER
CISCO CTAO OFFICE

BACKGROUND

- My name is Sape Mullender
- I worked at CWI from 1983 to 1990
- Guido van Rossum was a member of my team
- We worked on the Amoeba Distributed Operating System

- Later: Prof at the University of Twente (1990-1997, part-time thereafter), Director at Bell Laboratories (1997-2014), Chief Cook & Bottle Washer at Cisco (2014-now)

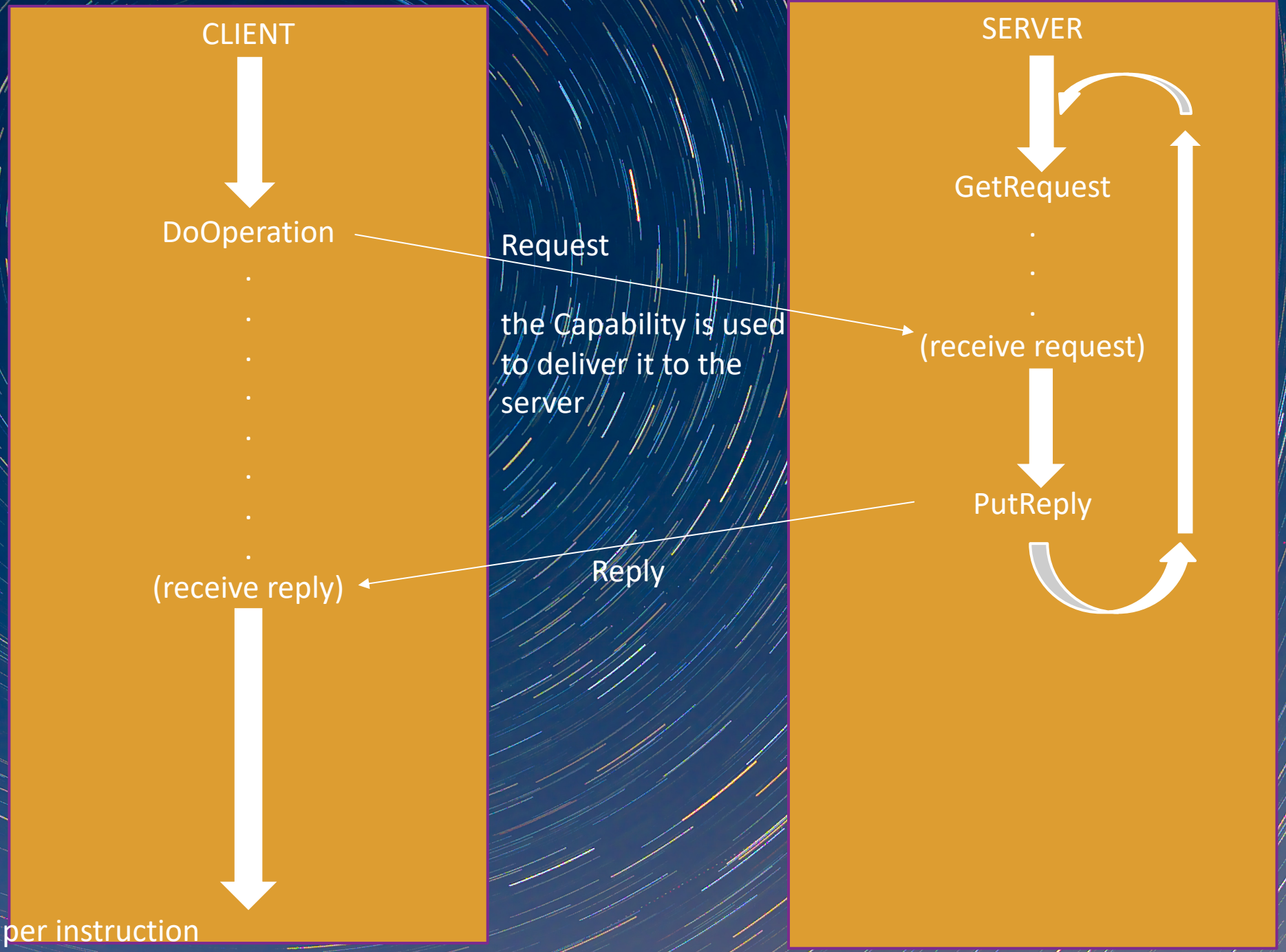
AMOEBA

All operations are “remote”

Client calls: DoOperation;
Server calls: GetRequest and then PutReply

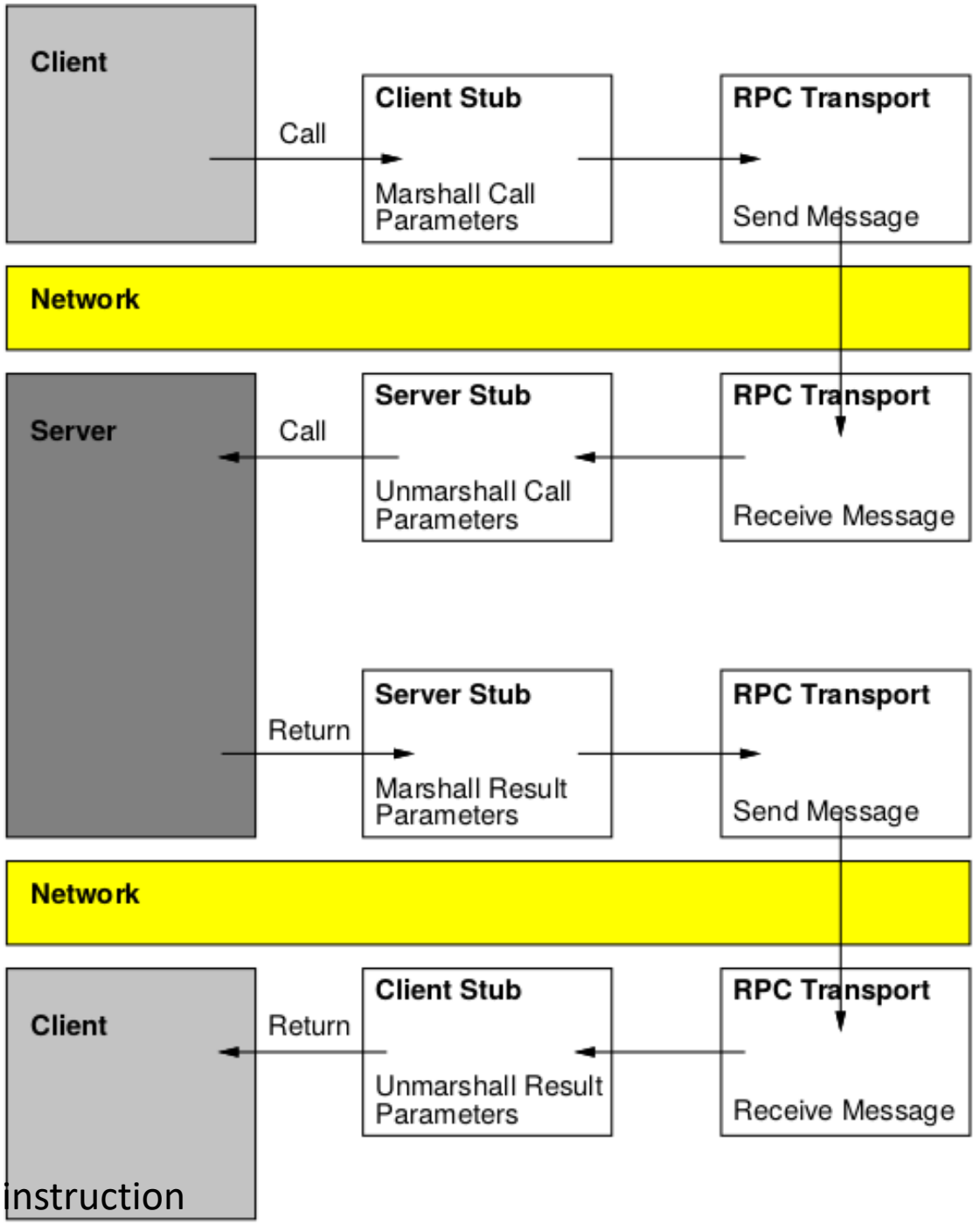
Names of services and objects are “secret”:
if you know the name, you can talk to it

The eighties: 1 µs per instruction



The eighties: 1 µs per instruction

Remote Procedure Call



The eighties: 1 µs per instruction

AIL - a Class-Oriented RPC Stub Generator for Amoeba

Guido van Rossum

CWI, Center for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
E-mail: guido@cwi.nl or mcvax!guido

ABSTRACT

AIL – an acronym for Amoeba Interface Language – is a class-oriented RPC stub generator, used with Amoeba's RPC primitives. Together with Amoeba's facilities for manipulating capabilities (bit patterns that are unforgeable references to objects maintained by servers anywhere on a network), AIL provides a completely object-oriented view of a distributed operating system.

Input to AIL consists of class and type definitions and generator directives; output are several files containing function definitions to be compiled and linked with clients and servers. Class definitions consist mainly of function headers (specifying parameter types, etc.). Classes can inherit multiple other classes. AIL can (in principle) generate stubs for different programming languages, so clients and servers need not be written in the same language.

ALTHOUGH AIL WAS BASED ON THE LANGUAGE C ..

Classes

The class concept in AIL differs quite a bit from that in C++. A class in AIL can contain only constant and type definitions and function prototypes; there are no data members. AIL classes specify only public information; there are no private definitions as in C++.

Since the client stubs are intended to be called by programs written in C or Pascal (for example), the class member call notation from C++ cannot be used to call client stubs generated by AIL. Instead of writing

```
object->member_function(argument1, argument2, ...)
```

the user must write

```
member_function(object, argument1, argument2, ...)
```

Class Inheritance

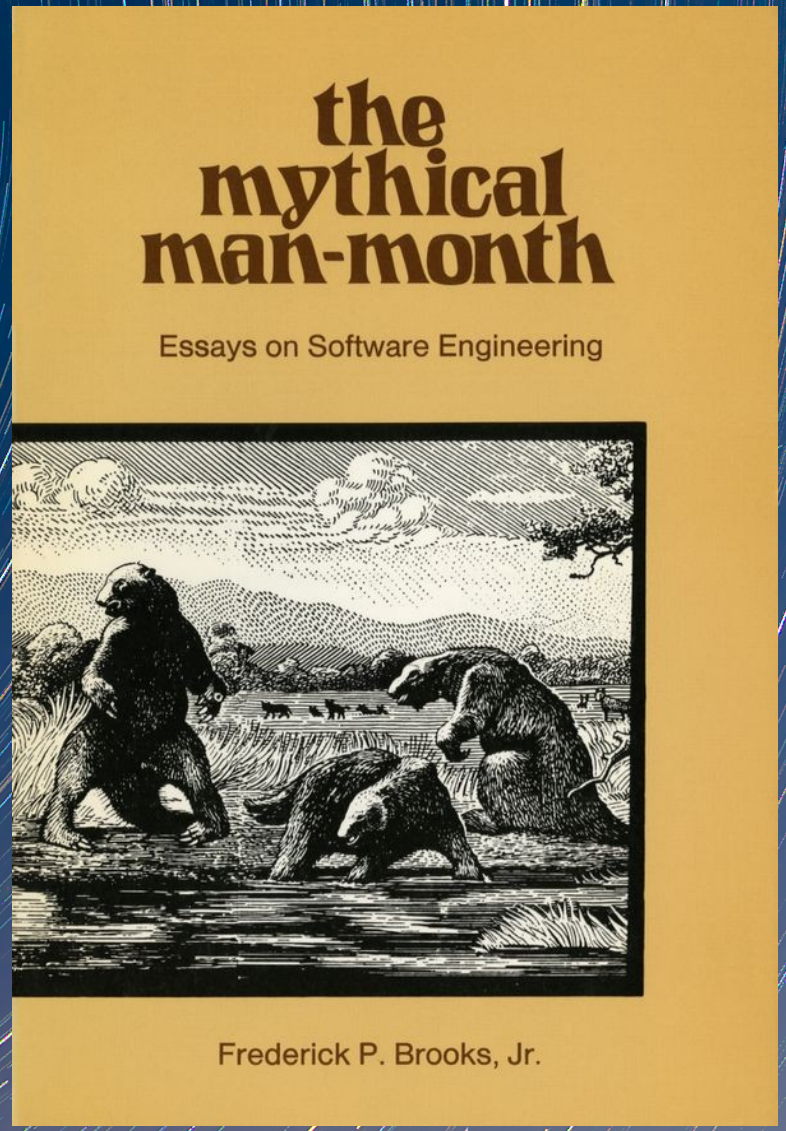
The power of the class mechanism lies in the possibility to extend existing classes by creating *derived classes*. A derived class has all the properties of its *base class(es)*, plus any properties added by its own definition. In C++, a derived class must be derived from exactly one base class; in AIL, a class can be derived from multiple base classes. This property is called *multiple inheritance*.

RPC WAS VERY POPULAR

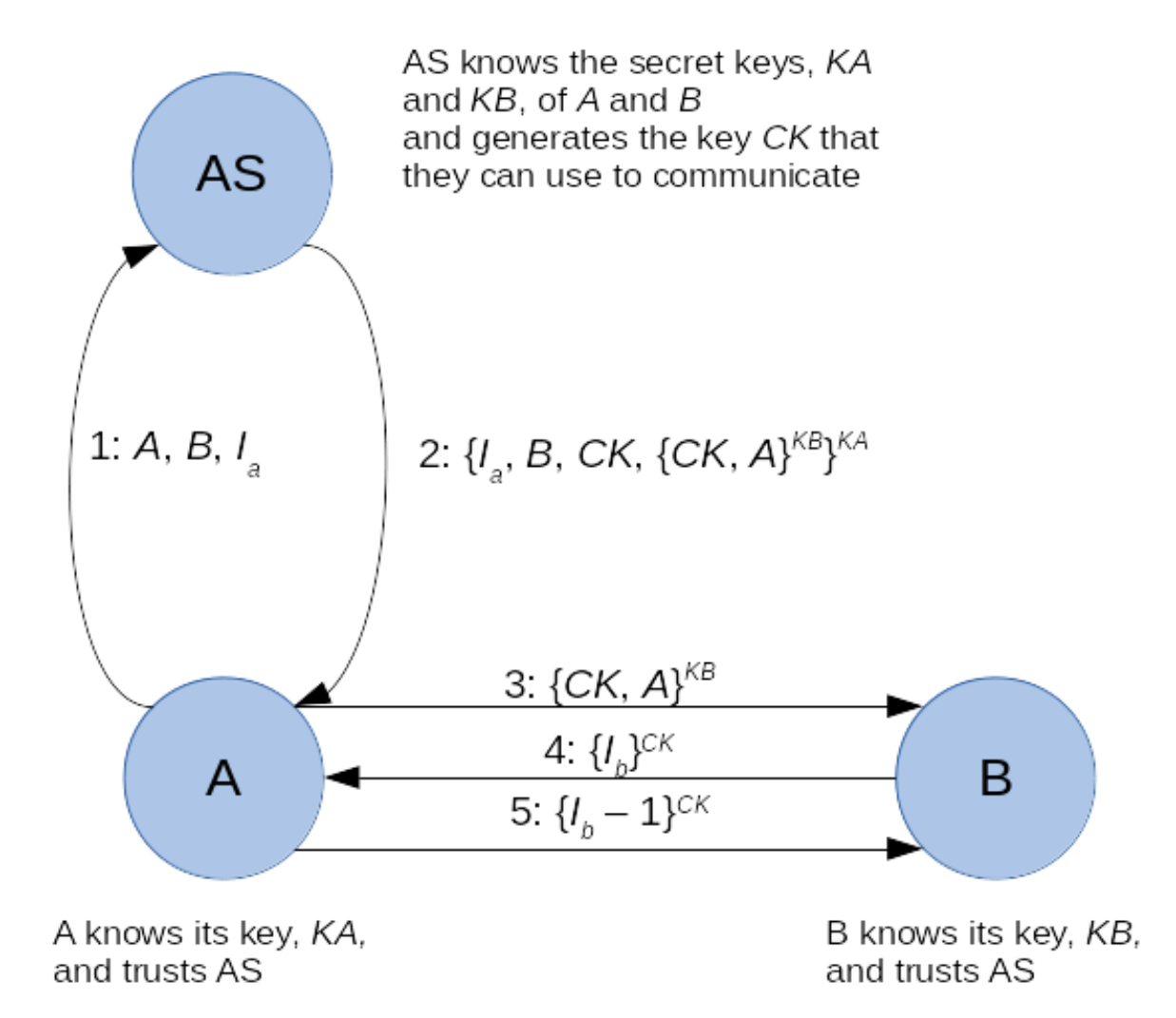
- So the industry set out to standardize it
- Rather than taking Guido's excellent design
- The industry came up with a much improved standard: Corba
- It was enthusiastically adopted
- But it wasn't used much: too complicated

THIS IS CALLED THE SECOND-SYSTEMS EFFECT

- Fred Brooks identified this in **The Mythical Man Month**
- (more than half a century old, but a Must Read for every systems designer)
- A first system works so well that the decision is made to “improve” it.
- The result is often too bloated and too complicated to work
- Or it misses the point completely …



THE NEEDHAM SCHROEDER PROTOCOL



- Five simple messages to let Alice (A) and Bob(B) create an aiuthenticated connection protected by key CK, with the help of the Authentication Server (AS)
- It had a flaw that wasn't discovered until almost 10 years later
- Alice can check that CK was generated moments ago (because message 2 contains the value I_a that Alice used in Message 1)
- But Bob has nothing to verify the "freshness" of the key

Authentication Revisited

R.M.Needham & M.D. Schroeder

In a paper published in 1978 (Needham & Schroeder) we presented protocols for the use of encryption for authentication in large networks of computers. Subsequently the protocols were criticised (Denning and Sacco) on the grounds that compromise of a session key and copying of an authenticator would enable an enemy to pretend indefinitely to be the originator of a secure conversation. In the notation of that paper, where A is the initiator and B is the other participant, A's first communication to B was:

A -> B $\{CK,A\}^{KB}$,

CK being the session key and KB being B's private key. If an enemy obtained CK and had copied the authenticator quoted above, he could always pretend to B that he was A. Denning and Sacco proposed a solution based on time-stamps, which we had rejected on the grounds that it required a good-quality time value to be universally available. Other possible solutions depended upon having state recorded in the authentication server, and more transactions with it.

In 1986 one of us (RMN) gave a lecture at the University of Tromsø which included the 1978 protocol, the criticism of it, and also a general principle about the use of nonce identifiers. This was that the identifier should always be generated by the party that sought reassurance about the time integrity of a transaction. In discussion Dr Sape J. Mullender of CWI Amsterdam pointed out that this should apply to the reassurance of B against the attack outlined. It may be achieved as follows:

A -> B A
B -> A $\{A,J\}^{KB}$, where J is a nonce identifier which will be kept by B

This quantity is passed by A to AS as an additional argument to A's opening call. The authentication server, knowing KB, decrypts it, checks that the partner name (A) matches the caller, and creates a slightly enhanced authenticator:

AS -> A $\{CK,A,J\}^{KB}$

On receipt of this, when decrypting to find CK, B can also check his nonce identifier and is thus protected.

Discussion

This proposal takes an extra interaction between A and B but requires no extra interactions with the authentication server and no accurate distributed clock - something that can only itself be maintained at the cost of interactions (see for example Lamport and Melliar-Smith). B has to maintain some extra state, but this is much better than a server doing so. Both A and B are assured of the freshness of the transaction - A because his interaction with the server is, in the complete protocol, protected by a nonce identifier, and B because of the use of J. One might suggest using J itself as the session key; we consider this to be dubious since if B is careless in his use of J (for example always using the same value) he leaves himself open to deception, whereas if CK is chosen in a similarly careless way both parties are vulnerable.

References

R.M. Needham & M.D. Schroeder, Using Encryption for Authentication in large Networks of Computers, CACM 21, 1978, p993

D.E. Denning & G.M. Sacco, Timestamps in Key Distribution Protocols, CACM 24, 1981, p533

L. Lamport & P.M. Melliar-Smith, Byzantine Clock Synchronization, Operating Systems Review 20-3, 1986, p10

THE BAN LOGIC

- When I was on sabbatical at the University of Cambridge in 1987, Mike Burrows, Martin Abadí, and Roger Needham were working on a Logic of Authentication to check protocols for flaws, the BAN Logic
- One of the protocols they analysed was the new standard for authenticatiuon, the X.509 Authentication Protocol
- It uses public key and it uses only 3 messages:

X.509

1. $A \rightarrow B: A, \{T_a, N_a, B, X_a, \{Y_a\}K_b\}K_a^{-1}$
2. $B \rightarrow A: B, \{T_b, N_b, A, N_a, X_b, \{Y_b\}K_a\}K_b^{-1}$
3. $A \rightarrow B: A, \{N_b\}K_a^{-1}$

- A encrypts Y_a with B's public key, then signs the message with its secret key
- Same for B

X.509

- But an intruder can remove the signature and replace it by its own
- So the authentication protocol does not necessarily *authenticate*
- Moreover, Messages 1 and 2 have identical formats, so, in a replay attack, a received message 2 could be sent as a fresh message 1 (exercise for the audience how you abuse this)

RESPONSE FROM THE X.509 STANDARDIZATION BODY

- After Roger Needham had alerted the committee in a private letter, the committee responded …
- “You may be right. Unfortunately, we are already too far into the standardization process to change it”

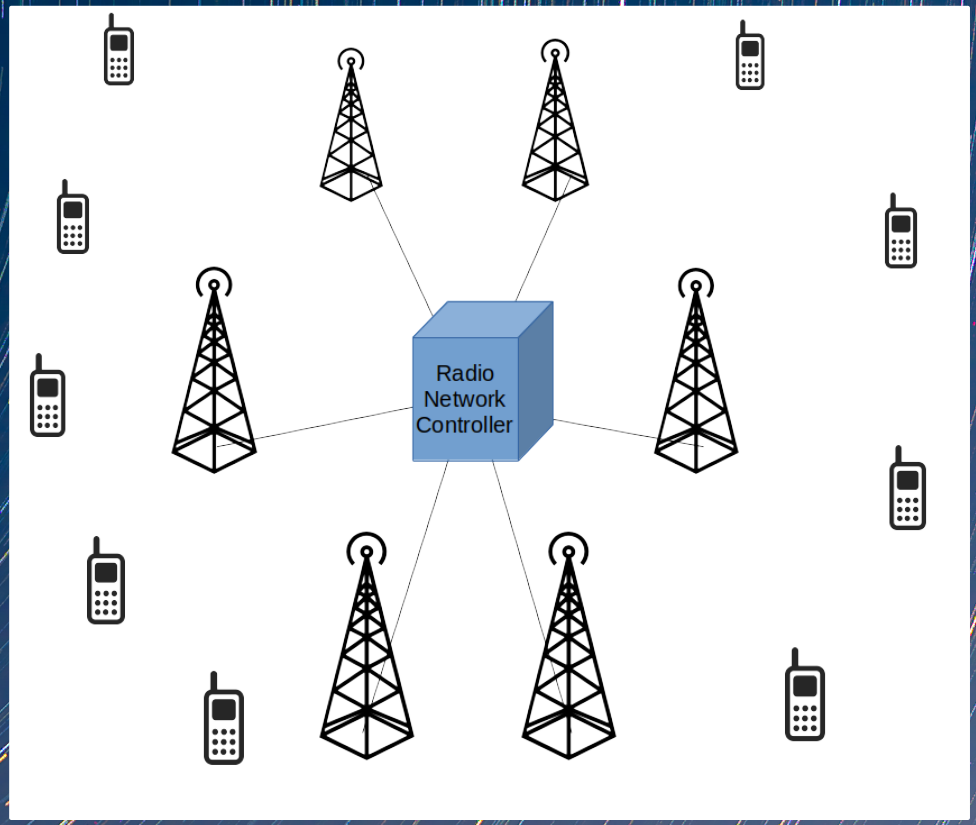
- This is the second-systems effect in all its glory

GSM → UMTS

- At Bell Labs in Murray Hill, I worked on UMTS, the “improved” second system after the very successful GSM
- Industry was very confident: The German government collected €49b for the auction of the spectrum; the UK government collected £22b: more than €600 per German or British citizen
- But UMTS was so complicated that the frequencies remained unused for more than a third of the 15-year lease

UMTS

- Peter Bosch and I proposed a design that eliminated the Radio Network Controller, an extremely expensive piece of equipment at the centre of the (GSM and) UMTS network, turning the base stations into a distributed system for managing calls and handovers.
- But the standard didn't allow it so it didn't happen until the next standard: 4G – and now 5G too



SECOND SYSTEMS

Almost always bad

- Algol60 → Algol68
- Unix → BSD
- C → C++
- IPv4 → IPv6
- Boeing 737 → Boeing 737-MAX
- ABC → Python

BENEVOLENT DICTATOR FOR LIFE

- Python is now the most popular programming language in introductory programming courses
- We have to be very grateful to Guido van Rossum for his vigilance in keeping the language elegant and clean
- His Benevolent Dictatorship has done this and it is an important contributing factor to today’s celebration

GUIDO VAN ROSSUM STEPPED DOWN AS BDFL

- But, in July 2018, Guido announced that he will be stepping down as BDFL
- What is this, Guido?
- You’re not dead yet!


Guido, don’t let Python fall prey to the Second-Systems Effect