The past, present, and futures of Ascon parameterisation

Arne Padmos

csrc.nist.gov

Lightweight Cryptography Standardization Process: NIST Selects Ascon

February 07, 2023

The NIST Lightweight Cryptography Team has reviewed the finalists based on their submission packages, status updates, third-party security analysis papers, and implementation and benchmarking results, as well as the feedback received during workshops and through the lwc-forum. The decision was challenging since most of the finalists exhibited performance advantages over NIST standards on various target platforms without introducing security concerns.

The team has decided to standardize the **Ascon** family for lightweight cryptography applications as it meets the needs of most use cases where lightweight cryptography is required. Congratulations to the Ascon team! NIST thanks all of the finalist teams and the community members who provided feedback that contributed to the selection.

NIST's next steps will be to:

- Publish NIST IR 8454, which describes the details of the selection and the evaluation process
- Work with the Ascon designers to draft the new lightweight cryptography standard for public comments
- Host a virtual public workshop to further explain the selection process and to discuss various aspects of standardization (e.g., additional variants, functionalities, and parameter selections) as well as possible extensions to the scope of the lightweight cryptography project. The tentative dates for the workshop are June 21-22, 2023. More information will be provided in the upcoming weeks.

NIST Lightweight Cryptography Team

Also see the related NIST news article, *NIST Selects 'Lightweight Cryptography' Algorithms to Protect Small Devices*.

RELATED TOPICS

Security and Privacy: lightweight cryptography

Activities and Products: standards development

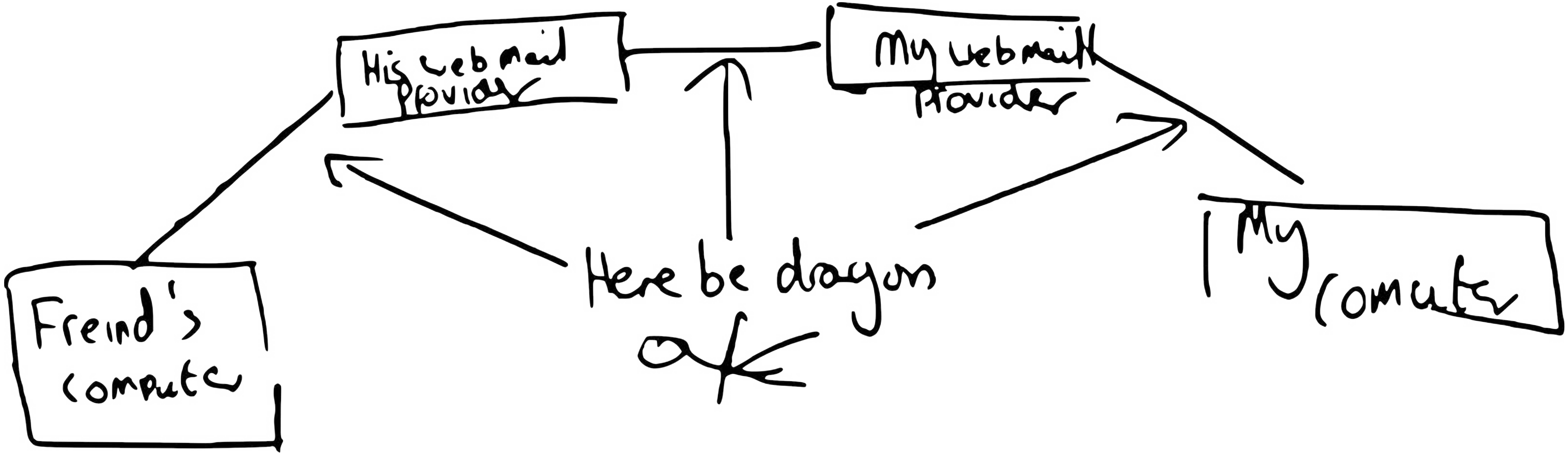
RELATED PAGES

News Item: Lightweight Cryptography Finalists Announced

Event: Lightweight Cryptography Workshop 2023

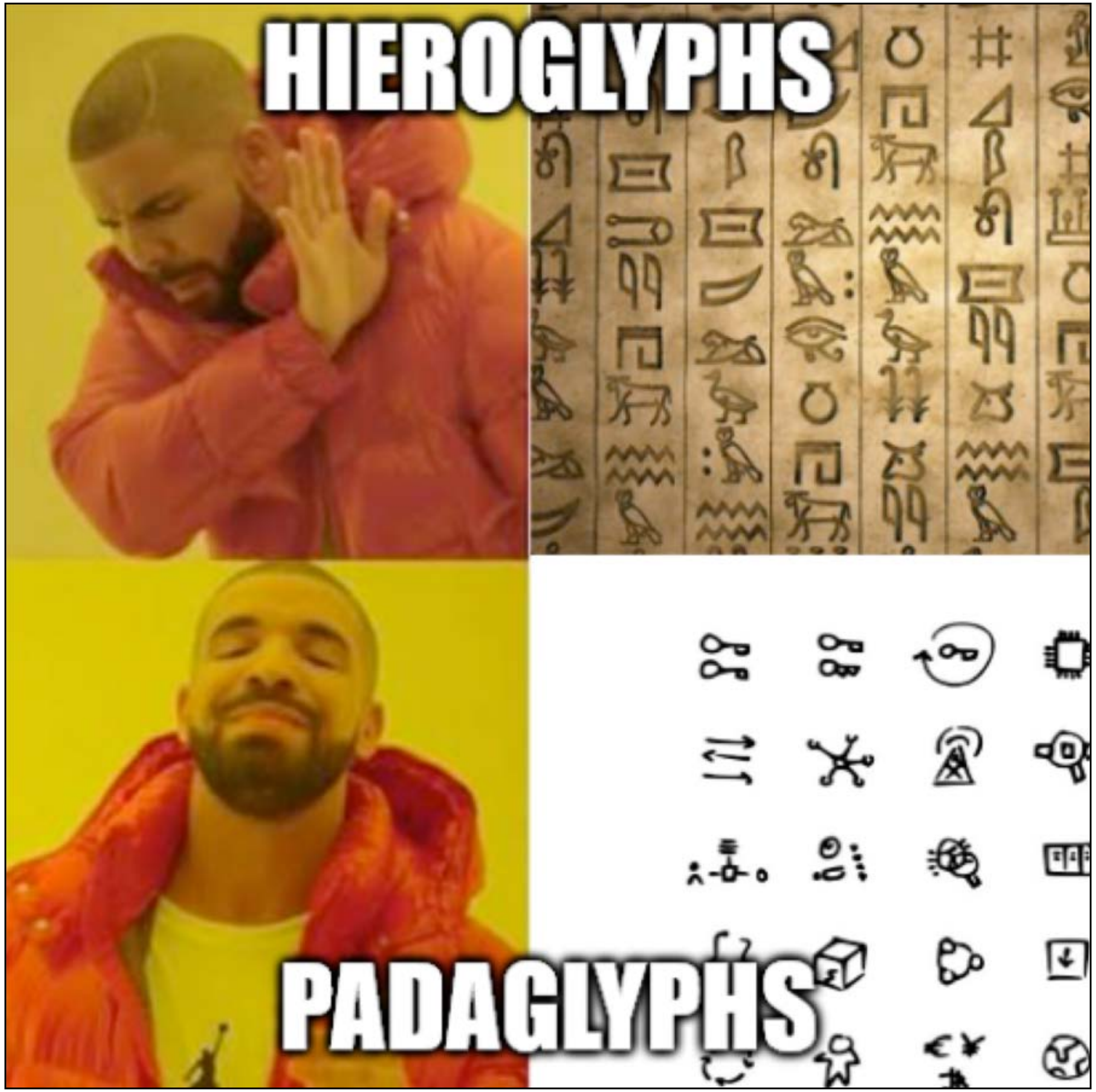
```
$ whoami
```

Ridley & Lawler, 2014

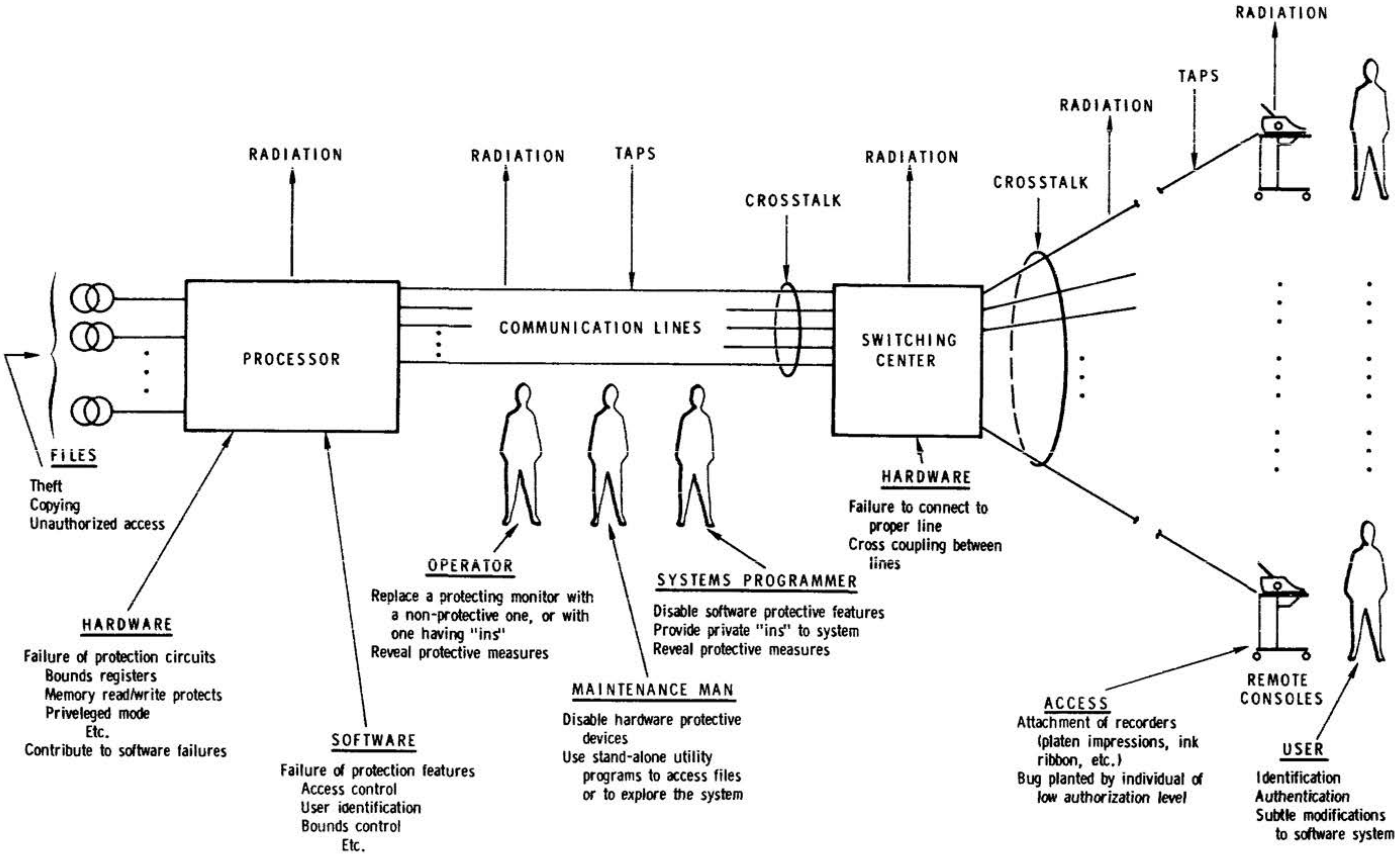


Renaud et al., 2014


ARDUINO
UNO



Ujinobu, 2020



Willis Ware, 1967

6
SEE SHEET Nº 8
SEE SHEET Nº 5
SEE SHEET Nº 2
SEE SHEET Nº 9
FLOATING HARBOUR
STONE BRIDGE
QUAY HEAD
QUAY STREET
ST STEPHEN STREET
ST LEONARDS LANE
BALDWIN STREET
ST NICHOLAS STREET
CORN STREET
SMALL STREET
BROAD STREET
ST JOHNS STREET
OLD POST OFFICE PASSAGE
SHANNON COURT
BACK HALL STEPS
ENT. STA. HALL
CITY SUPPLY STORES
NATIONAL PROVINCIAL BANK
TIMES & MIRROR PRINTING OFFICE
WATERWORKS
COUNTY COURT
ST WERBURGHS CHAMBERS
LEWIS BROS PROVISION WHSE
J.W. ARROWSMITH PRINTER
WALSH & Cº CLOTH WHSE
WINE & SPIRIT MERCHANT OFFICES
POST OFFICE
WAREHOUSE
WHSE WINE & SPIRIT
COMMERCIAL ROOMS
INSURANCE BUILDINGS
LONDON & SOUTH WESTERN BANK OFFICES
STUCKEY'S BANK
SOMERSET CHAMBERS
IMPERIAL BLDGS.
LIVERPOOL LONDON AND GLOBE INS. OFFICE
LECTURE HALL
J. WHITE & SONS WINE & SPIRIT VAULTS
READING ROOM
BANK CHAMBERS
AUCTION MART
WESTERN CHAMBERS
FLOUR WHSE
FISH MARKET
RESTAURANT
TEA WHSE
WINE & SPIRIT CELLARS
POULTRY MARKET 1ST
PRINTERS OVER
NORTHERN ASSCE OFFICES
EXCHANGE BUILDINGS
CORN EXCHANGE
GLASS & IRON ROOF
MEAT MARKET
VEGETABLE MARKET
FRUIT
AUCTION ROOMS
ALBION CHAMBERS
ALLIANCE CHAMBERS
BRISTOL & WEST OF ENGLAND BANK
BANK OF ENGLAND
GUILDHALL
COURT ROOM
LIBRARY
BROAD ST HALL SALE ROOMS
PUBLIC HOUSE
50
51
53
54
56

National Cyber Security Centre
Ministry of Justice and Security

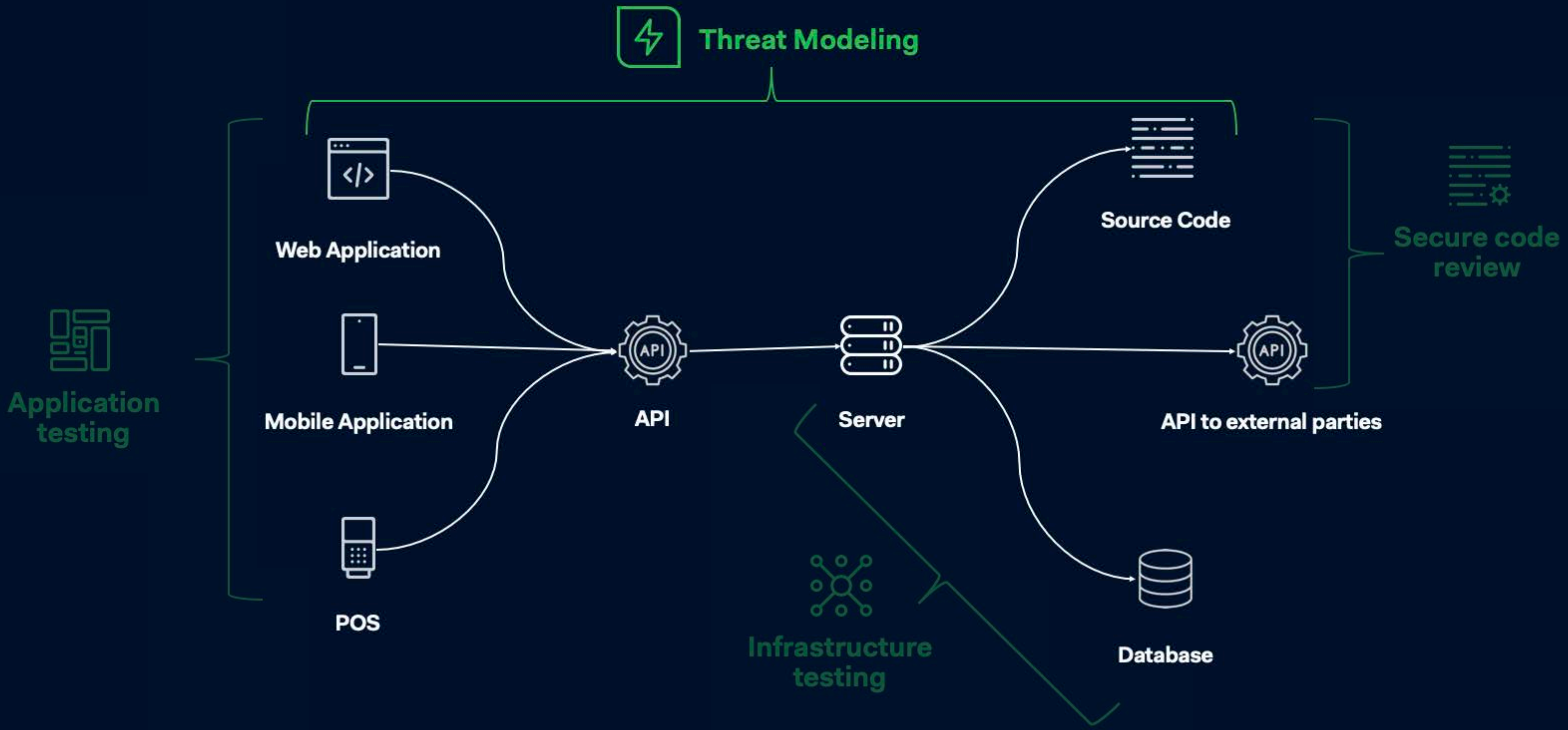
Guidelines for quantum-safe transport-layer encryption

These guidelines are written for an audience of architects responsible for specifying cryptographic requirements. They can also be used in R&D and prototyping as well as for contract negotiations. For a more general introduction, see NLNCSA's brochure and our own factsheet. For further details, follow NIST, ETSI, IETF, and ISO standardisation efforts and read publications by ENISA and TNO.

Our recommendations target the early adopters who follow our advice to apply quantum-safe cryptography to ensure long-term confidentiality against store-and-decrypt attacks. Signatures are not part of these guidelines as they are not vulnerable to such attacks. The guidelines recommend hybrid key exchange to mitigate potential vulnerabilities in novel post-quantum algorithms and implementations. Besides a list of algorithms and recommended parameters, this document also contains some questions to ask when choosing implementations.

Combine traditional algorithms with quantum-safe key encapsulation

Key agreement should rely on multiple algorithms. For other purposes, apply established methods. You should use algorithms that have stood the test of time and that are future-proof. However, post-quantum cryptography is a new and fast-moving field. As such, ensure that you can quickly replace any algorithms and implementations that you rely on – so-called cryptographic agility.


Threat Modeling
Web Application
Mobile Application
POS
API
Server
Source Code
API to external parties
Database
Application testing
Secure code review
Infrastructure testing

Disclaimer:
all opinions are my own

Cryptographic competitions

An illustrated history of Ascon

Real-world challenges

Lessons from usable security

Back to the future of PBC

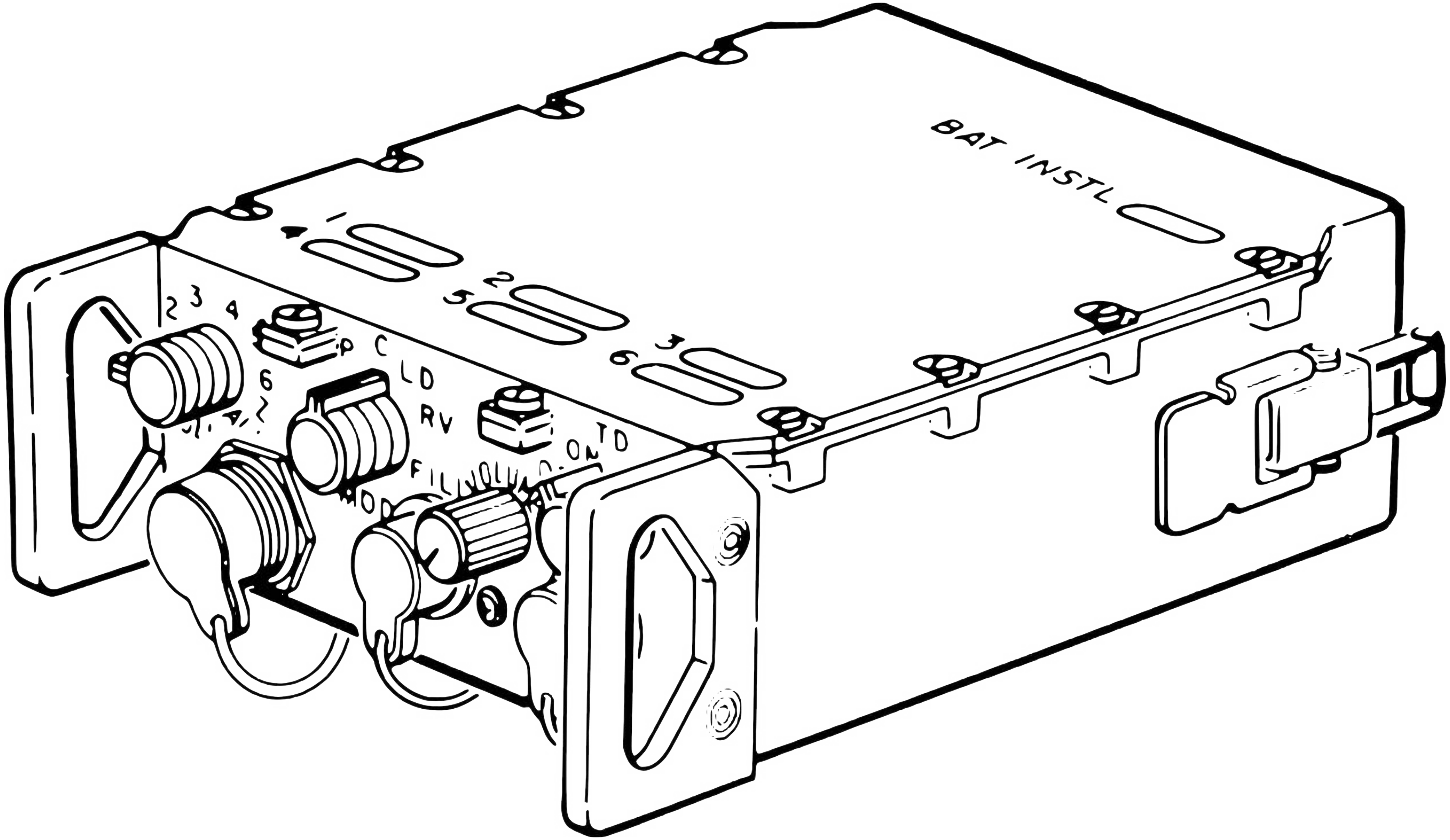
Cryptographic competitions

An illustrated history of Ascon

Real-world challenges

Lessons from usable security

Back to the future of PBC



Department of the Army, 1985

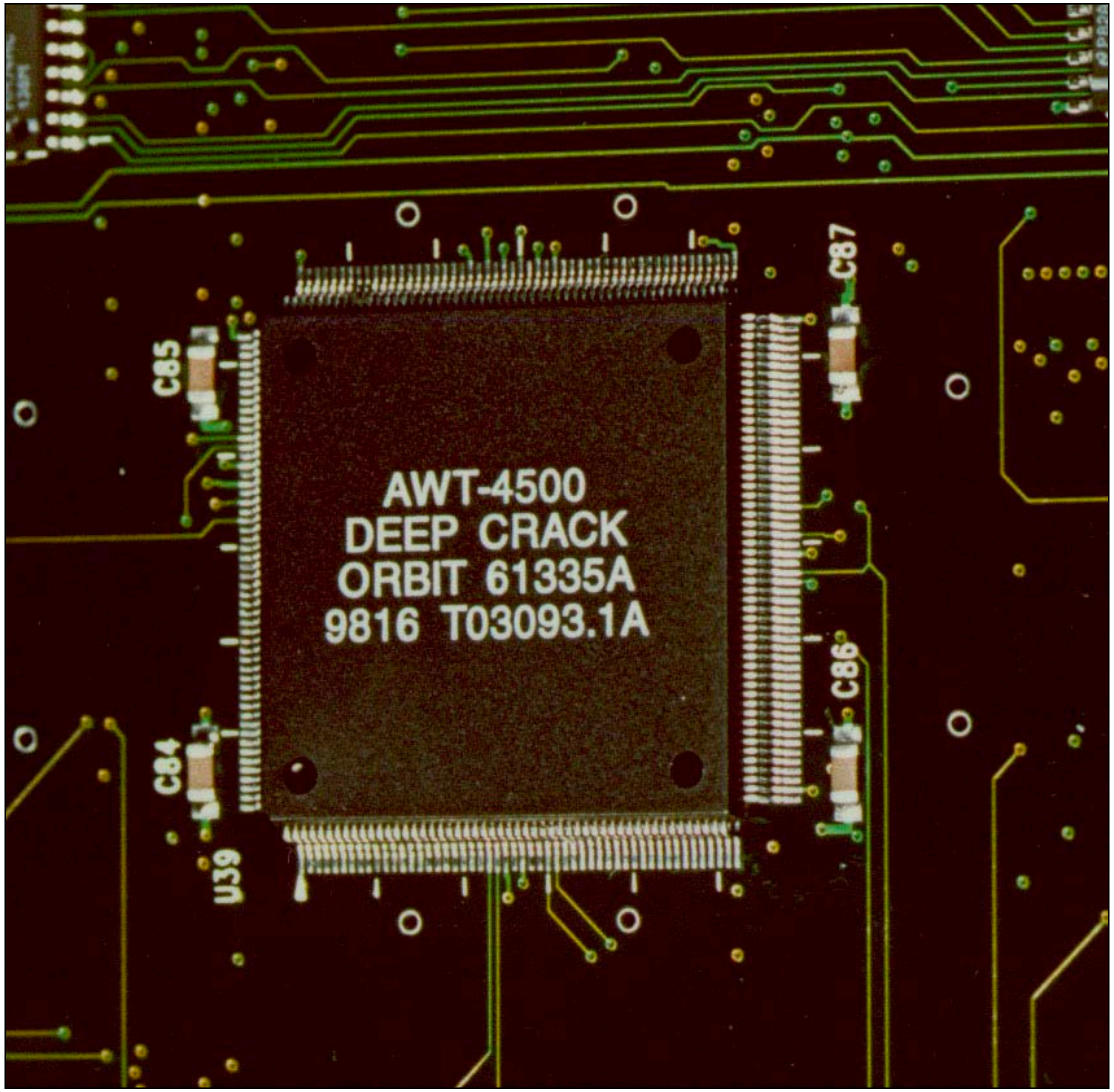
FIPS PUB 46

FEDERAL INFORMATION
PROCESSING STANDARDS PUBLICATION
1977 JANUARY 15

U.S. DEPARTMENT OF COMMERCE / National Bureau of Standards

DATA ENCRYPTION STANDARD

CATEGORY: ADP OPERATIONS
SUBCATEGORY: COMPUTER SECURITY


AWT-4500
DEEP CRACK
ORBIT 61335A
9816 T03093.1A
C87
C85
C86
C84
U39

EFF, 1998


AES
CAST-256
CRYPTON
DEAL
DFC
LOKI97
MAGENTA
MARS
RC6
RIJNDAEL
SAFER+
AES Efficiency Testing

NIST, 1999

Cache-timing attacks on AES

Daniel J. Bernstein *

Department of Mathematics, Statistics, and Computer Science (M/C 249)
The University of Illinois at Chicago
Chicago, IL 60607–7045
djb@cr.yp.to

Abstract. This paper demonstrates complete AES key recovery from known-plaintext timings of a network server on another computer. This attack should be blamed on the AES design, not on the particular AES library used by the server; it is extremely difficult to write constant-time high-speed AES software for common general-purpose computers. This paper discusses several of the obstacles in detail.

Keywords: side channels, timing attacks, software timing attacks, cache timing, load timing, array lookups, S-boxes, AES

1 Introduction

This paper reports successful extraction of a complete AES key from a network server on another computer. The targeted server used its key solely to encrypt data using the OpenSSL AES implementation on a Pentium III.

The successful attack was a very simple timing attack. Presumably the same technique can extract complete AES keys from the more complicated servers actually used to handle Internet data, although the attacks will often require

Volume 126, Article No. 126024 (2021) https://doi.org/10.6028/jres.126.024
Journal of Research of the National Institute of Standards and Technology

Development of the Advanced Encryption Standard

Miles E. Smid

Formerly: Computer Security Division,
National Institute of Standards and Technology,
Gaithersburg, MD 20899, USA

mesmid@verizon.net

Strong cryptographic algorithms are essential for the protection of stored and transmitted data throughout the world. This publication discusses the development of Federal Information Processing Standards Publication (FIPS) 197, which specifies a cryptographic algorithm known as the Advanced Encryption Standard (AES). The AES was the result of a cooperative multiyear effort involving the U.S. government, industry, and the academic community. Several difficult problems that had to be resolved during the standard's development are discussed, and the eventual solutions are presented. The author writes from his viewpoint as former leader of the Security Technology Group and later as acting director of the Computer Security Division at the National Institute of Standards and Technology, where he was responsible for the AES development.

Key words: Advanced Encryption Standard (AES); consensus process; cryptography; Data Encryption Standard (DES); security requirements, SKIPJACK.

Accepted: June 18, 2021

Published: August 16, 2021; **Current Version:** August 23, 2021

This article was sponsored by James Foti, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology (NIST). The views expressed represent those of the author and not necessarily those of NIST.

https://doi.org/10.6028/jres.126.024

1. Introduction

In the late 1990s, the National Institute of Standards and Technology (NIST) was about to decide if it was going to specify a new cryptographic algorithm standard for the protection of U.S. government and commercial data. The current standard was showing signs of age and would not be up to the task of providing strong security much longer. NIST could step aside and let some other entity manage the development of new cryptographic standards, it could propose a short-term fix with a limited lifetime, or it could establish a procedure to develop a completely new algorithm. In January 1997, NIST decided to move forward with a proposal for developing an Advanced Encryption Standard (AES), which would be secure enough to last well into the next millennium. In December of 2001, after five years of effort, the finished standard was approved and published. The journey from initial concept to final standard was not straightforward. This paper covers the motivation for the development of the AES, the process that was followed, and the problems that were encountered and solved along the way. It documents a significant milestone in the history of NIST's computer security program, which will be celebrating its 50th anniversary in 2022.

How to cite this article:
Smid ME (2021) Development of the Advanced Encryption Standard.
J Res Natl Inst Stan 126:126024. https://doi.org/10.6028/jres.126.024

NISTIR 8319

Review of the Advanced Encryption Standard

Nicky Mouha

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8319

National Institute of Standards and Technology
U.S. Department of Commerce

Guidelines for Submissions of Modes of Operation

Submissions should specify a mode of operation for a symmetric (secret) key block cipher algorithm. At a minimum, the mode should support underlying block ciphers with key-block combinations of 128-128, 192-128, and 256-128 bits. However, the specification should be generic – i.e., written to handle other key-block combinations, if they can be supported. Example modes include, but are not limited to, techniques for performing encryption, message authentication, hashing, and random bit generation. It will be helpful to receive variations of Counter mode arising from alternative methods/guidelines for prescribing the generation of counters.

NIST requests that submissions of modes of operation include the following six items:

- cover sheet
- mode specification
- summary of properties
- test vectors
- performance estimates
- intellectual property statements/agreements/disclosures.

These items are discussed below.

Cover Sheet

The cover sheet shall contain the following information:

- name of submitted mode of operation;
- principal submitter's name, telephone, fax, organization, postal address, e-mail address;
- name(s) of auxiliary submitter(s);
- name of mode's inventor(s)/developer(s);
- name of owner, if any, of the mode (typically, the owner will be the same as the submitter).

Mode Specification

A complete written specification of the mode of operation should be provided, including all mathematical equations, tables, diagrams, and parameters that are needed to implement the mode. NIST encourages submitters to elaborate on the intended use(s) of the mode, the design rationale, the relevant properties, proofs (if any), the comparison with other modes, and the mode's overall advantages/disadvantages.

Summary of Properties

Morris Dworkin, 2001

To assist NIST and the public to draw comparisons and contrasts between the various candidate modes, the submissions should include a table or outline that identifies the following characteristics:

4 JULY 2001

DUAL COUNTER MODE

MIKE BOYLE

CHRIS SALTER

INTRODUCTION

For the past 18 months, the NSA has been developing a high-speed encryption mode for IP packets. The mode that we designed is identical in many aspects to Jutla's Integrity Aware Parallelizable Mode (IAPM). There is one important difference in our proposal. In the IP world, a large number of packets might arrive out of order. Integrity Aware Parallelizable Mode (IAPM) and the proposed variations incur a large overhead for out of order packets[JU 01]. Each packet requires at least the time to perform a full decryption to obtain an IV before decryption of the cipher can begin. This note describes our solution to this problem.

First, we describe the basic mode and its features. We then describe how to implement this mode for IPSec.

DUAL COUNTER MODE

Dual counter mode is a hybrid of ECB mode and counter mode. Let E represent encryption by a codebook of width W. Let P_1, P_2, ..., P_j be j blocks of plaintext and let C_1, C_2, ..., C_j be the corresponding ciphertext. Let f be a polynomial of degree W for a primitive linear feedback shift register. Also, let $\{x_i\}$ be the sequence of fills generated by this polynomial. The first fill, x_0, is a secret shared between the two peers. This initial fill is most easily derived from the key exchange[1]. Dual counter mode can be described as follows:

j = # of datablocks

For i = 1, ..., j

$$x_i = f(x_{i-1})$$

$$C_i = E(P_i \oplus x_i) \oplus x_i$$

Quite likely the cipherblocks will travel in packets. If the packets arrive in order, the receiver does not lose track of the fill needed to decrypt the cipher.

TWO IMPLEMENTATION MODES

We knew that many implementers would want to verify the data integrity of packets. This mode has the property that any change to a ciphertext block causes the decrypted plaintext to be garbled. Thus it is easy to add a checksum to verify data integrity.

[1] Of course, care should be taken in producing this value. For example, the designers of the key exchange for IPsec used secure hashes such as SHA-1 to isolate keying material.

A Note on NSA's Dual Counter Mode of Encryption

Pompiliu Donescu * pompiliu@eng.umd.edu
Virgil D. Gligor ** gligor@eng.umd.edu
David Wagner *** daw@cs.berkeley.edu

September 28, 2001

Abstract. We show that both variants of the Dual Counter Mode of encryption (DCM) submitted for consideration as an AES mode of operation to NIST by M. Boyle and C. Salter of the NSA are insecure with respect to both secrecy and integrity in the face of chosen-plaintext attacks. We argue that DCM cannot be easily changed to satisfy its stated performance goal and be secure. Hence repairing DCM does not appear worthwhile.

1 Introduction

On August 1, 2001, M. Boyle and C. Salter of the NSA submitted two variants of the Dual Counter Mode (DCM) of encryption [1] for consideration as an AES mode of operation to NIST. The DCM goals are: (1) to protect both the secrecy and integrity of IP packets (as this mode is intended to satisfy the security goals of Jutla's IAPM mode [4]), and (2) to avoid the delay required before commencing the decryption of out-of-order IP packets, thereby decreasing the decryption latency of IAPM. DCM is also intended to allow high rates of encryption.

The authors argue that DCM satisfies the first goal because "an error in a cipher block causes all data in the packet to fail the integrity check". DCM appears to satisfy the second goal because it maintains a "shared secret negotiated during the key exchange," which avoids the delay inherent to the decryption of a secret IV before the first out-of-order packet arrival can be decrypted. The authors note correctly that Jutla's IAPM mode does not satisfy their second goal.

In this note, we show that both variants of DCM are insecure with respect to both secrecy and integrity in the face of chosen-plaintext attacks. Further, we argue that DCM cannot be easily changed to satisfy its stated performance goal for the decryption of out-of-order packets *and* be secure. We conclude since other proposed AES modes satisfy the proposed goals for DCM, even if repairing DCM is possible, which we doubt, such an exercise does not appear to be worthwhile.

[1] VDG Inc., 6009 Brookside Drive, Chevy Chase, MD 20815.
[2] Electrical and Computer Engineering Department, University of Maryland, College Park, Maryland 20742.
[3] Computer Science Division, EECS Department, University of California Berkeley, Berkeley, CA. 94720.

Cryptanalysis of OCB2

Akiko Inoue and Kazuhiko Minematsu

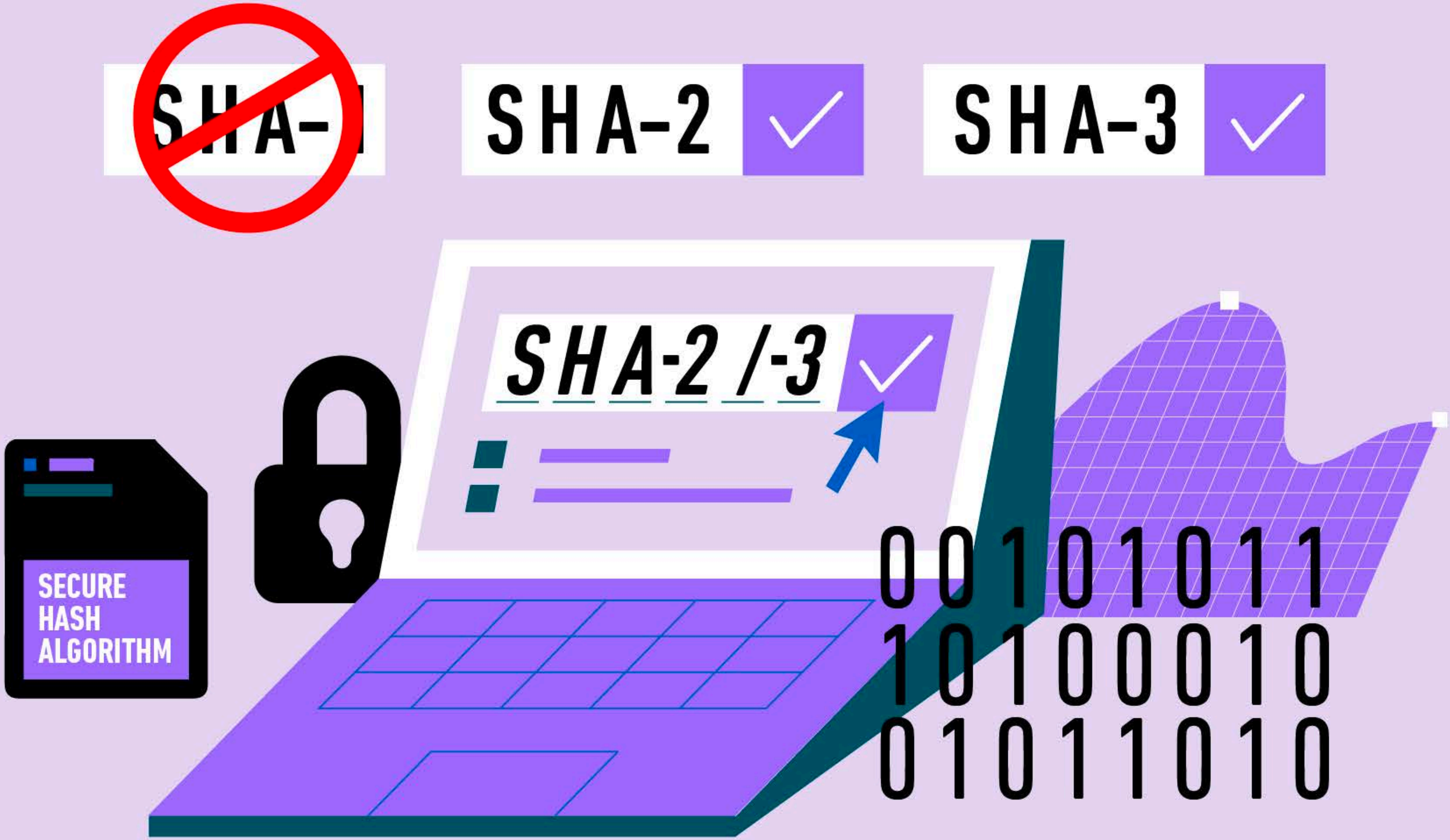
NEC Corporation, Japan
a-inoue@cj.jp.nec.com, k-minematsu@ah.jp.nec.com

Abstract. We present practical attacks against OCB2, an ISO-standard authenticated encryption (AE) scheme. OCB2 is a highly-efficient blockcipher mode of operation. It has been extensively studied and widely believed to be secure thanks to the provable security proofs. Our attacks allow the adversary to create forgeries with single encryption query of almost-known plaintext. This attack can be further extended to powerful almost-universal and universal forgeries using more queries. The source of our attacks is the way OCB2 implements AE using a tweakable blockcipher, called XEX^*. We have verified our attacks using a reference code of OCB2. Our attacks do not break the privacy of OCB2, and are not applicable to the others, including OCB1 and OCB3.

Keywords: OCB, Authenticated Encryption, Cryptanalysis, Forgery, XEX

1 Introduction

Authenticated encryption (AE) is a form of symmetric-key encryption that provides both confidentiality and authenticity of messages. Now it is widely accepted


SHA-2
SHA-3
SHA-2 /-3
SECURE
HASH
ALGORITHM
00101011
10100010
01011010

NIST, 2022

A Vulnerability in Implementations of SHA-3, SHAKE, EdDSA, and Other NIST-Approved Algorithms

Nicky Mouha[1(✉)][0000−0001−8861−782X] and Christopher Celi[2][0000−0001−9979−6819]

[1] Strativia, Largo, MD, USA
nicky@mouha.be
[2] National Institute of Standards and Technology, Gaithersburg, MD, USA
christopher.celi@nist.gov

Abstract. This paper describes a vulnerability in several implementations of the Secure Hash Algorithm 3 (SHA-3) that have been released by its designers. The vulnerability has been present since the final-round update of Keccak was submitted to the National Institute of Standards and Technology (NIST) SHA-3 hash function competition in January 2011, and is present in the eXtended Keccak Code Package (XKCP) of the Keccak team. It affects all software projects that have integrated this code, such as the scripting languages Python and PHP Hypertext Preprocessor (PHP). The vulnerability is a *buffer overflow* that allows attacker-controlled values to be eXclusive-ORed (XORed) into memory (without any restrictions on values to be XORed and even far beyond the location of the original buffer), thereby making many standard protection measures against buffer overflows (e.g., canary values) completely ineffective. First, we provide Python and PHP scripts that cause segmentation faults when vulnerable versions of the interpreters are used. Then, we show how this vulnerability can be used to construct second preimages

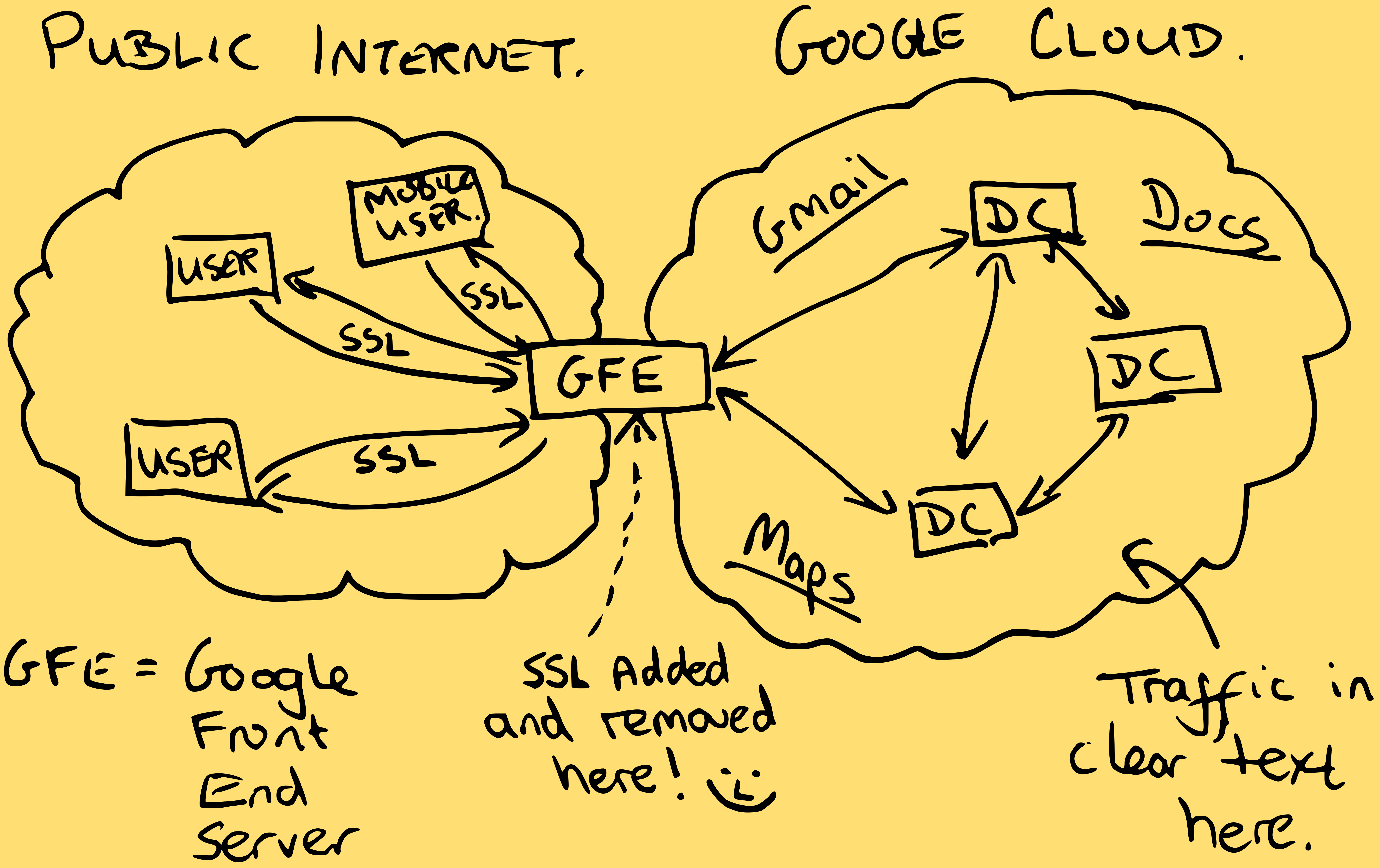
***Bla*KE12**

We are proud to announce

***Bla*zing-fast**
KECCAK on
12 rounds.

***Bla*KE12** (/ˈbleɪki: twelv/), or KECCAK reduced to 12 rounds, is a blazing-fast cryptographic hash function with a rock-solid security foundation and suitable to a wide-range of platforms.

Keccak team, 2022



Gellman & Soltani, 2013

Snowden Disclosures

- News stories came out strongly suggesting that Dual EC had a trapdoor inserted by NSA
- This put the previous discussions in an entirely new light.
- We responded by:
    - Issuing an ITL bulletin telling everyone to stop using Dual EC DRBG until further notice.
    - Putting all three 800-90 documents up for public comment

NIST Cryptographic Standards and Guidelines Development Process

Report and Recommendations of the
Visiting Committee on Advanced Technology
of the National Institute of Standards and Technology

July 2014

VCAT

NISTIR 7977

NIST Cryptographic Standards and Guidelines Development Process

Cryptographic Technology Group

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.IR.7977

NIST
National Institute of
Standards and Technology
U.S. Department of Commerce

PHC Lessons Learned

Algorithm competitions, when parameterised appropriately, work

- Collaborative evolution of new crypto mechanisms

Can be run in complete openness

- No need for behind-closed-doors deliberations or government intervention

Dealing with hypothetical but practically irrelevant weaknesses is a problem when the cost to mitigate is significant

- Damned if you do, damned if you don't

Cryptographic competitions

Daniel J. Bernstein[1,2]

[1] Department of Computer Science, University of Illinois at Chicago,
Chicago, IL 60607–7045, USA
[2] Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
djb@cr.yp.to

Abstract. Competitions are widely viewed as the safest way to select cryptographic algorithms. This paper surveys procedures that have been used in cryptographic competitions, and analyzes the extent to which those procedures reduce security risks.

Keywords: cryptography, competitions, DES, AES, eSTREAM, SHA-3, CAESAR, NISTPQC, NISTLWC

1 Introduction

> *The CoV individual reports point out several shortcomings and procedural weaknesses that led to the inclusion of the Dual EC DRBG algorithm in SP 800-90 and propose several steps to remedy them. . . . The VCAT strongly encourages standard development through open*

Research

Assurance

— HACKATHON CHALLENGE —

BRIDGING THE GAP BETWEEN MAKING AND BREAKING

Arne Padmos

Context. Compared to the popularity of both hackathons and CTF challenges, as well as the impact that AES has had and that NIST's PQC selection is expected to have, very little research has been done on what we call, for lack of an established term, adversarial engineering design competitions. This is unfortunate, as such competitions appear to be a useful tool for assured technology transfer. Given that NIST will review their guidelines for cryptographic standards development this year, it would be opportune as a WEIS community to explore and provide insights into how the shape of competitions can influence incentives and drive assurance.

Challenge. Can we use competitions to improve the state of security, and if so, how might we structure competitions to include both defensive and offensive aspects in order to bridge the divide between the making and breaking of computer systems?

Concept. Competitions that focus on breaking stuff are a common occurrence at many security conferences, reflecting our field's focus on looking for problems

Cryptographic competitions

An illustrated history of Ascon

Real-world challenges

Lessons from usable security

Back to the future of PBC

Grain - A Stream Cipher for Constrained Environments

Martin Hell[1], Thomas Johansson[1] and Willi Meier[2]

[1] Dept. of Information Technology, Lund University,
P.O. Box 118, 221 00 Lund, Sweden
{martin,thomas}@it.lth.se
[2] FH Aargau, CH-5210 Windisch, Switzerland
meierw@fh-aargau.ch

Abstract. A new stream cipher, Grain, is proposed. The design targets hardware environments where gate count, power consumption and memory is very limited. It is based on two shift registers and a nonlinear output function. The cipher has the additional feature that the speed can be increased at the expense of extra hardware. The key size is 80 bits and no attack faster than exhaustive key search has been identified. The hardware complexity and throughput compares favourably to other hardware oriented stream ciphers like E0 and A5/1.

1 Motivation

When designing a cryptographic primitive there are many different properties that have to be addressed. These include e.g. speed, security and simplicity. Comparing several ciphers, it is likely that one is faster on a 32 bit processor, another is faster on an 8 bit processor and yet another one is faster in hardware. The simplicity of the design is another factor that has to be taken into account, but while the software implementation can be very simple, the hardware implementation might be quite complex.

There is a need for cryptographic primitives that have very low hardware complexity. An RFID tag is a typical example of a product where the amount of memory and power is very limited. These are microchips capable of transmitting an identifying sequence upon a request from a reader. Forging an RFID tag can have devastating consequences if the tag is used e.g. in electronic payments and hence, there is a need for cryptographic primitives implemented in these tags. Today, a hardware implementation of e.g. AES on an RFID tag is not feasible due to the large number of gates needed. Grain is a stream cipher primitive that is designed to be very easy and small to implement in hardware.

Many stream ciphers are based on linear feedback shift registers (LFSR), not only for the good statistical properties of the sequences they produce, but also for the simplicity and speed of their hardware implementation. Several recent LFSR based stream cipher proposals, see e.g. [6, 7] and their predecessors, are based on word oriented LFSRs. This allows them to be efficiently implemented in software

PRESENT: An Ultra-Lightweight Block Cipher

A. Bogdanov[1], L.R. Knudsen[2], G. Leander[1], C. Paar[1], A. Poschmann[1], M.J.B. Robshaw[3], Y. Seurin[3], and C. Vikkelsoe[2]

[1] Horst-Görtz-Institute for IT-Security, Ruhr-University Bochum, Germany
[2] Technical University Denmark, DK-2800 Kgs. Lyngby, Denmark
[3] France Telecom R&D, Issy les Moulineaux, France
leander@rub.de, {abogdanov,cpaar,poschmann}@crypto.rub.de
lars@ramkilde.com, chv@mat.dtu.dk
{matt.robshaw,yannick.seurin}@orange-ftgroup.com

Abstract. With the establishment of the AES the need for new block ciphers has been greatly diminished; for almost all block cipher applications the AES is an excellent and preferred choice. However, despite recent implementation advances, the AES is not suitable for extremely constrained environments such as RFID tags and sensor networks. In this paper we describe an ultra-lightweight block cipher, PRESENT. Both security and hardware efficiency have been equally important during the design of the cipher and at 1570 GE, the hardware requirements for PRESENT are competitive with today's leading compact stream ciphers.

1 Introduction

One defining trend of this century's IT landscape will be the extensive deployment of tiny computing devices. Not only will these devices feature routinely in consumer items, but they will form an integral part of a pervasive — and unseen — communication infrastructure. It is already recognized that such deployments bring a range of very particular security risks. Yet at the same time the cryptographic solutions, and particularly the cryptographic primitives, we have at hand are unsatisfactory for extremely resource-constrained environments.

In this paper we propose a new hardware-optimized block cipher that has been carefully designed with area and power constraints uppermost in our mind. Yet, at the same time, we have tried to avoid a compromise in security. In achieving this we have looked back at the pioneering work embodied in the DES [34] and complemented this with features from the AES finalist candidate Serpent [4] which demonstrated excellent performance in hardware.

At this point it would be reasonable to ask why we might want to design a new block cipher. After all, it has become an "accepted" fact that stream ciphers are, potentially, more compact. Indeed, renewed efforts to understand the design of compact stream ciphers are underway with the eSTREAM [15] project and several promising proposals offer appealing performance profiles. But we note a couple of reasons why we might want to consider a compact block cipher. First, a block cipher is a versatile primitive and by running a block cipher in *counter*

What’s needed in the IoT era is not more Kirtland’s warblers and koalas, as wonderful as such animals may be, but **crows and coyotes**. An animal that eats only eucalyptus leaves, even if it outcompetes the koala, will never become widely distributed.

National Security Agency, 2015

The Simon and Speck Families of Lightweight Block Ciphers

Ray Beaulieu
Douglas Shors
Jason Smith
Stefan Treatman-Clark
Bryan Weeks
Louis Wingers

National Security Agency
9800 Savage Road, Fort Meade, MD 20755, USA

{rabeaul, djshors, jksmit3, sgtreat, beweeks, lrwinge}@tycho.ncsc.mil

19 June 2013

Abstract

In this paper we propose two families of block ciphers, Simon and Speck, each of which comes in a variety of widths and key sizes. While many lightweight block ciphers exist, most were designed to perform well on a single platform and were not meant to provide high performance across a range of devices. The aim of Simon and Speck is to fill the need for secure, flexible, and analyzable lightweight block ciphers. Each offers excellent performance on hardware and software platforms, is flexible enough to admit a variety of implementations on a given platform, and is amenable to analysis using existing techniques. Both

Chapter 4
An Account of the ISO/IEC Standardization of the Simon and Speck Block Cipher Families

Tomer Ashur and Atul Luykx

Abstract Simon and Speck are two block cipher families published in 2013 by the US National Security Agency (NSA). These block ciphers, targeting lightweight applications, were suggested in 2015 to be included in *ISO/IEC 29192-2 Information technology—Security techniques—Lightweight cryptography—Part 2: Block ciphers*. Following 3.5 years of deliberations within ISO/IEC JTC 1 they were rejected in April 2018. This chapter provides an account of the ISO/IEC standardization process for Simon and Speck.

4.1 Introduction

By their very nature, cryptographic algorithms require large-scale agreement to enable secure communication. Standardization by bodies such as ANSI, IEEE, and ISO/IEC is important means by which industries and governments achieve

NISTIR 8114

Report on Lightweight Cryptography

Kerry A. McKay
Larry Bassham
Meltem Sönmez Turan
Nicky Mouha

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8114

National Institute of Standards and Technology
U.S. Department of Commerce

NIST Cybersecurity White Paper *csrc.nist.gov*

Profiles for the Lightweight Cryptography Standardization Process

Larry Bassham
Çağdaş Çalık
Kerry McKay
Nicky Mouha
Meltem Sönmez Turan

Computer Security Division
Information Technology Laboratory

April 26, 2017

National Institute of Standards and Technology
U.S. Department of Commerce

Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process

Table of contents

NISTIR 8268

Status Report on the First Round of the NIST Lightweight Cryptography Standardization Process

Meltem Sönmez Turan
Kerry A. McKay
Çağdaş Çalık
Donghoon Chang
Larry Bassham

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8268

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 8369

Status Report on the Second Round of the NIST Lightweight Cryptography Standardization Process

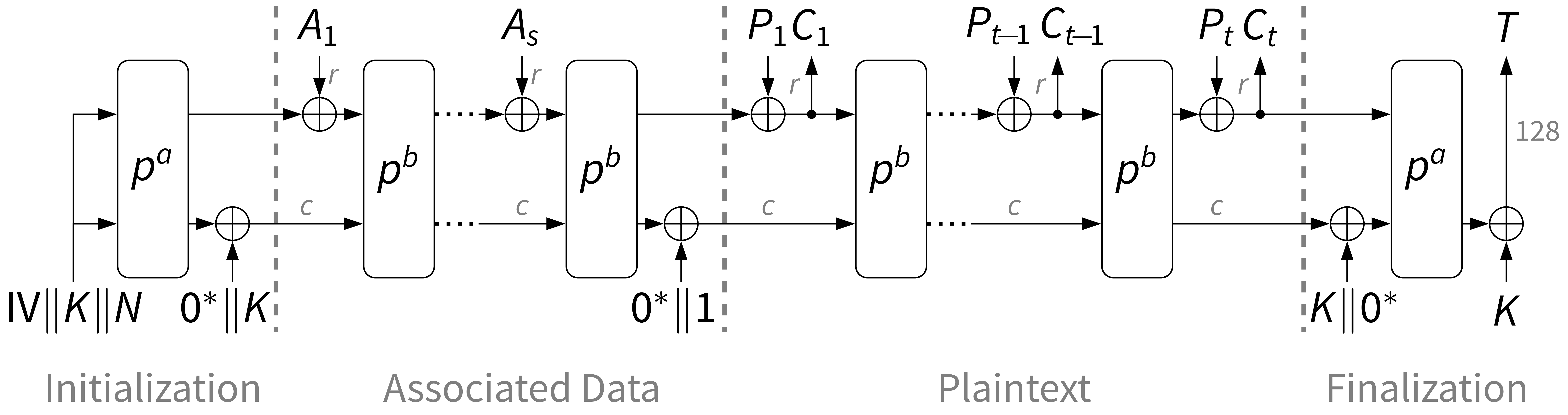
Meltem Sönmez Turan
Kerry McKay
Donghoon Chang
Çağdaş Çalık
Lawrence Bassham
Jinkeon Kang
John Kelsey

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8369

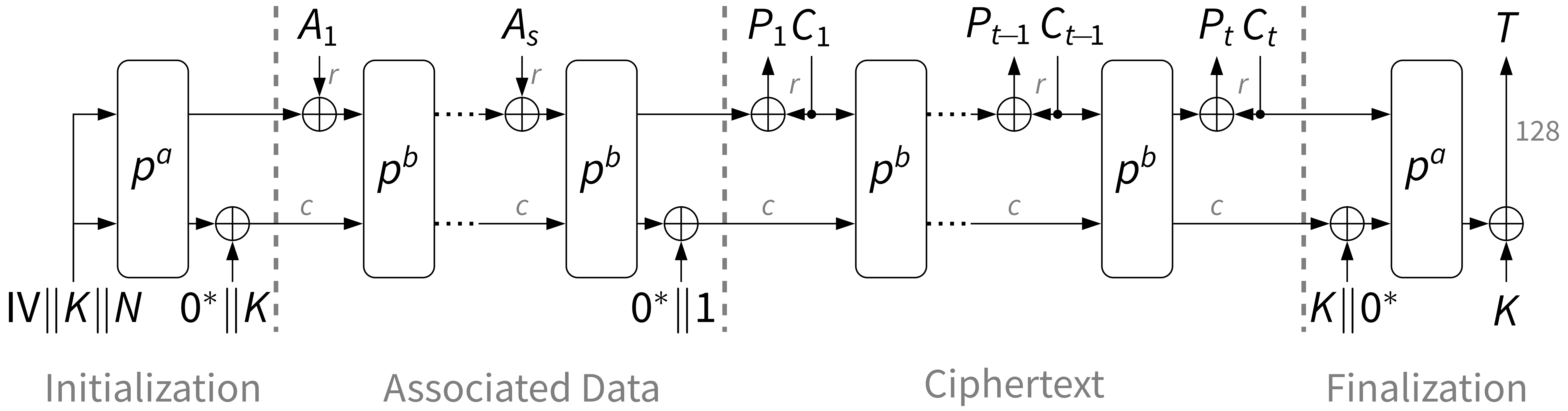
National Institute of Standards and Technology
U.S. Department of Commerce

?

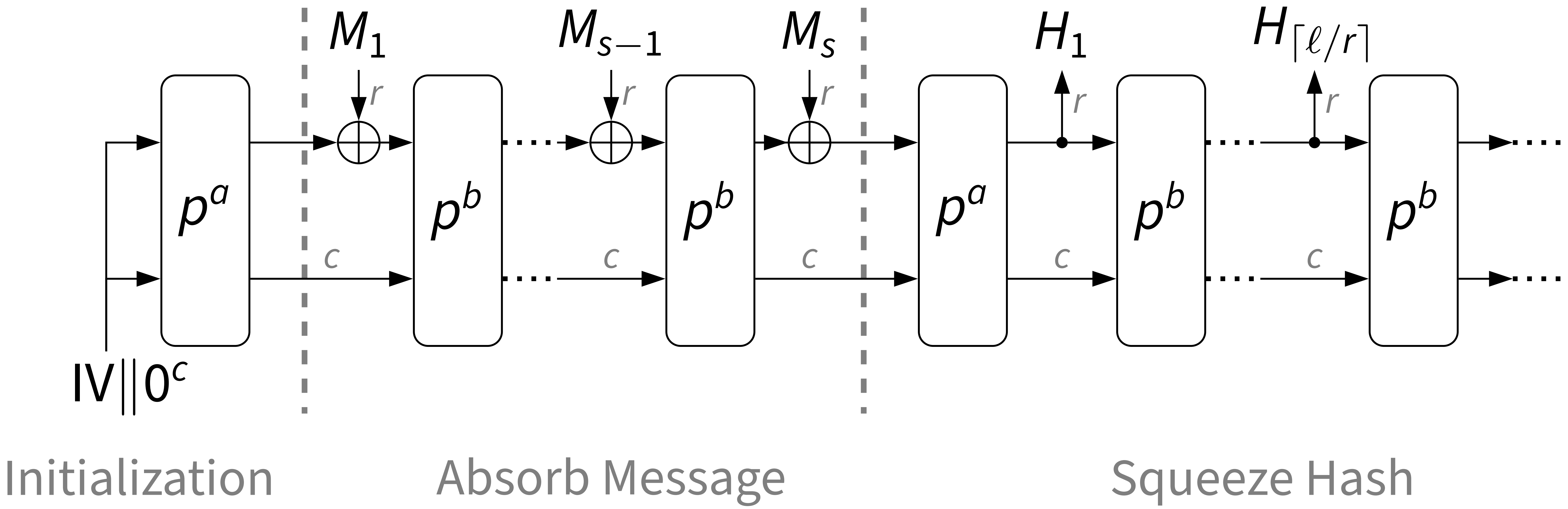
Helmut Lunghammer, 2023



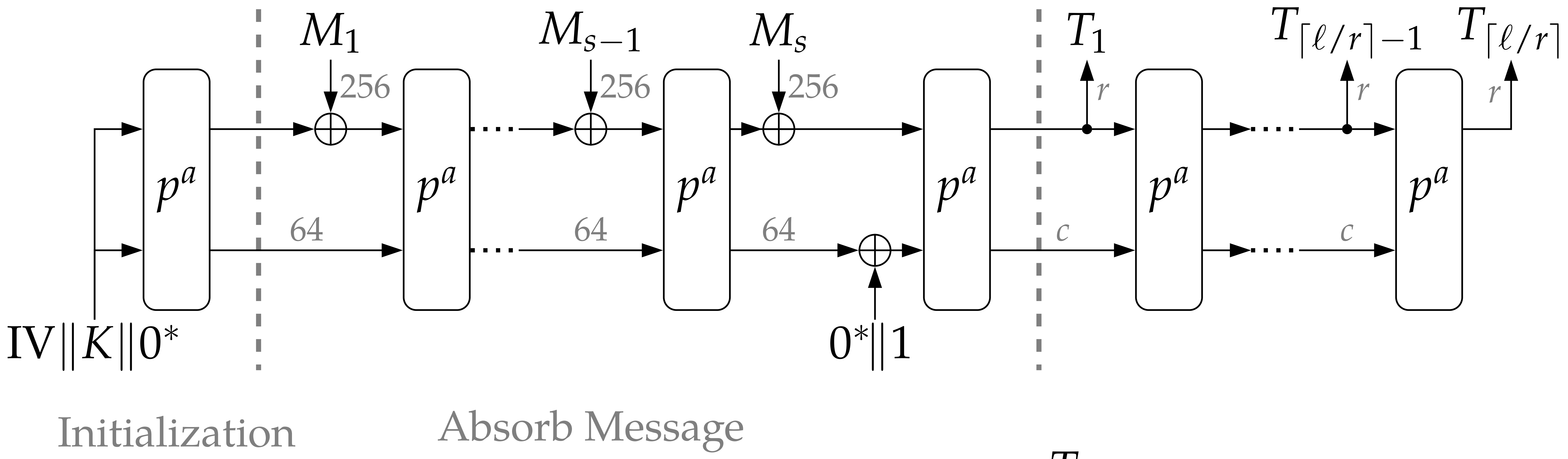
Dobraunig et al., 2021

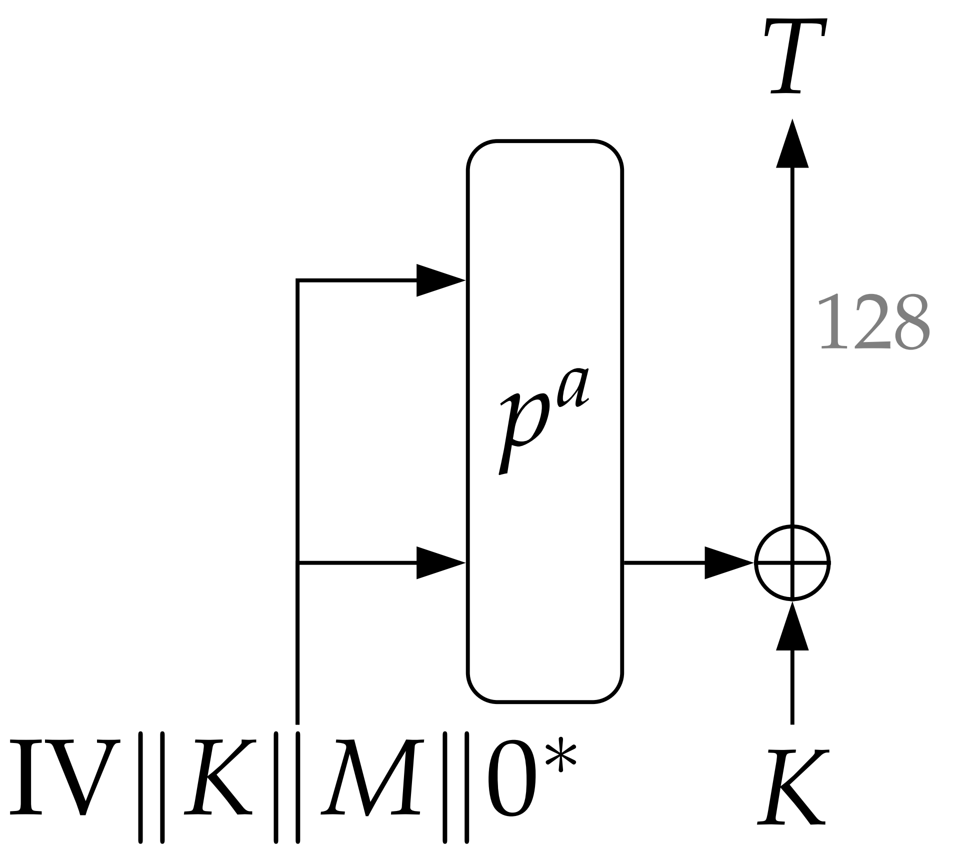


Dobraunig et al., 2021



Dobraunig et al., 2021





Dobraunig et al., 2021

Algorithms

This is a simple reference implementation of Ascon v1.2 as submitted to the NIST LWC competition that includes

- Authenticated encryption `ascon_encrypt(key, nonce, associateddata, plaintext, variant="Ascon-128")` (and similarly `decrypt`) with the following 3 family members:
  - `Ascon-128`
  - `Ascon-128a`
  - `Ascon-80pq`
- Hashing algorithms `ascon_hash(message, variant="Ascon-Hash", hashlength=32)` including 4 hash function variants with fixed 256-bit (`Hash`) or variable (`Xof`) output lengths:
  - `Ascon-Hash`
  - `Ascon-Hasha`
  - `Ascon-Xof`
  - `Ascon-Xofa`
- Message authentication codes `ascon_mac(key, message, variant="Ascon-Mac", taglength=16)` including 5 MAC variants (from https://eprint.iacr.org/2021/1574, not part of the LWC proposal) with fixed 128-bit (`Mac`) or variable (`Prf`) output lengths, including a variant for short messages of up to 128 bits (`PrfShort`).
  - `Ascon-Mac`
  - `Ascon-Maca`
  - `Ascon-Prf`
  - `Ascon-Prfa`
  - `Ascon-PrfShort`

Finalist	# Variants	Key size (bits)	Nonce size (bits)	Tag size (bits)	Digest size (bits)
Ascon	2 AEAD 2 hash	128 --	128 --	128 --	-- 256
Elephant	3 AEAD	128	96	64-128	--
GIFT-COFB	1 AEAD	128	128	128	--
Grain-128aead	1 AEAD	128	96	64	--
ISAP	4 AEAD	128	128	128	--
PHOTON-Beetle	2 AEAD 1 hash	128 --	128 --	128 --	-- 256
Romulus	3 AEAD 1 hash	128 --	128 --	128 --	-- 256
Sparkle	4 AEAD 2 hash	128-256 --	128-256 --	128-256 --	-- 256-384
TinyJambu	3 AEAD	128-256	96	64	
Xoodyak	1 AEAD 1 hash	128 --	128 --	128 --	-- 256

The Selection Process

- Fair evaluation of finalists is challenging.
  - Assigning weights for different evaluation criteria (security, performance in software and hardware, design maturity, amount of third-party analysis, IP issues, etc.)
  - Different security claims, different functionality, attacks with different complexities etc.
  - Limited resources (not all algorithms got the same attention from the crypto community)
- Decision relied on publicly available analysis and benchmarking results.
- In February 2023, NIST announced the Ascon family as the winner.
  - Large amount of third-party analysis
  - AEAD variants were listed part of the CAESAR portfolio for ‘lightweight applications’.
  - No tweak
  - Performance advantage over NIST standards in software and hardware

lightweight devices communicate with lightweight devices, but also for scenarios where many lightweight devices communicate with high-end devices (e.g., a back-end server), a typical use case in many applications including the Internet of Things (IoT). This is especially true in scenarios where protection against side-channel attacks is needed.

4 Planned tweak proposals

We do not plan any tweaks for Ascon.

Acknowledgments. The authors would like to thank all researchers contributing to the design, analysis and implementation of Ascon. In particular, we want to thank Hannes Gross and Robert Primas for all their support and various implementations of Ascon.

Part of this work has been supported by the Austrian Science Fund (FWF): P26494-N15 and J 4277-N38.

References

[1] A. Adomnicai, J. J. Fournier, and L. Masson. "Masking the Lightweight Authenticated Ciphers ACORN and Ascon in Software". Cryptology ePrint Archive,

Dobraunig et al., 2020

- Added a new hash function Ascon-Hasha and extendable output function Ascon-Xofa to the Ascon familiy.

  Compared to Ascon-Hash and Ascon-Xof, Ascon-Hasha and Ascon-Xofa use 8 rounds during absorbing and most of the squeezing instead of 12, while the transition between absorbing and squeezing still uses 12 rounds. We have reduced the number of rounds where the current analysis shows a very large security margin in order to get a less conservative and faster variant that pairs nicely with Ascon-128a. Moreover, we hope that these less conservative variants Ascon-Hasha and Ascon-Xofa encourage more cryptanalysis of the hash function in the last round of the standardization process.

`asconv12.pdf`:

- Updated Chapter 1 to introduce also the new variants Ascon-Hasha and Ascon-Xofa
- Replaced the algorithm $\mathcal{X}_{h,r,a}$ with $\mathcal{X}_{h,r,a,b}$ in order to define new variants Ascon-Hasha and Ascon-Xofa in Chapter 2. $\mathcal{X}_{h,r,a,b}$ is identical to $\mathcal{X}_{h,r,a}$ if $a = b$ and so $\mathcal{X}_{h,r,a,a} = \mathcal{X}_{h,r,a}$.
- Added Ascon-Hasha to the recommended parameter sets at second place for hash function in Section 2.2.
- Added Ascon-128a and Ascon-Hasha as recommended pairing for authenticated encryption and hashing in Section 2.2.

Dobraunig et al., 2021

The recommendation for NIST includes Ascon-Hash combined with Ascon-128 or Ascon-128a.

— Ascon-Hash AND (Ascon-128 OR Ascon-128a)
— Ascon-Hash AND (Ascon-128 XOR Ascon-128a)
— (Ascon-Hash AND Ascon-128) OR Ascon-128a
— (Ascon-Hash AND Ascon-128) XOR Ascon-128a

Next Steps

NIST

- Publication of the third–round status update

- Sixth Lightweight Cryptography Workshop in June 21-22 2023 (virtual)

  Submission deadline: May 1, 2023

  Aim: to explain the selection process, and to discuss various aspects of lightweight cryptography standardization, such as
  - Which ASCON variants to standardize? All of subset ? XOF instead of hash?
  - Additionally functionality, e.g. dedicated MAC?
  - Support for additional parameter sizes? e.g., larger nonce, shorter tags

- Publication of draft standard (in 2023)

Competition	CAESAR			NIST LWC		
Algorithm	V1 15-03-2014	V1.1 29-08-2015	V1.2 15-09-2016	V1.2 29-03-2019	V1.2 27-09-2019	V1.2 17-05-2021
Ascon-128	800c0600 00000000	80400c06 00000000	80400c06 00000000	80400c06 00000000	80400c06 00000000	80400c06 00000000
Ascon-128a	—	80800c08 00000000	80800c08 00000000	80800c08 00000000	80800c08 00000000	80800c08 00000000
Ascon-96	600c0800 00000000	—	—	—	—	—
Ascon-80pq	—	—	—	a0400c06 xxxxxxxx	a0400c06 xxxxxxxx	a0400c06 xxxxxxxx
Ascon-Hash	—	—	—	00400c00 00000100	00400c00 00000100	00400c00 00000100
Ascon-Hasha	—	—	—	—	—	00400c04 00000100
Ascon-XOF	—	—	—	00400c00 00000000	00400c00 00000000	00400c00 00000000
Ascon-XOFa	—	—	—	—	—	00400c04 00000000

Competition	**CAESAR**			**NIST LWC**		
Algorithm	v1	v1.1	v1.2	v1.2	v1.2	v1.2
Ascon-MAC	—	—	—	—	—	—
Ascon-MACa	—	—	—	—	—	—
Ascon-PRF	—	—	—	—	—	—
Ascon-PRFa	—	—	—	—	—	—
Ascon-PRFshort	—	—	—	—	—	—

Algorithm	IACR 03-12-2021	GitHub 21-09-2022	GitHub 24-03-2023
Ascon-MAC	80808c00 xxxxxxxx	80808c00 00000080	80808c00 00000080
Ascon-MACa	—	80808c04 00000080	80808c04 00000080
Ascon-PRF	80808c00 xxxxxxxx	80808c00 00000000	80808c00 00000000
Ascon-PRFa	—	80808c04 00000000	80808c04 00000000
Ascon-PRFshort	80xx4cxx 00000000	80xx4c80 00000000	80xx4cxx 00000000

```c
int crypto_prf(unsigned char* out, unsigned long long outlen,
               const unsigned char* in, unsigned long long inlen,
               const unsigned char* k) {
  if (inlen > 16 || outlen > 16 || outlen > CRYPTO_BYTES) return -1;
  /* load key */
  const uint64_t K0 = LOADBYTES(k, 8);
  const uint64_t K1 = LOADBYTES(k + 8, 8);
  /* initialize */
  ascon_state_t s;
  s.x[0] = ASCON_PRFS_IV | (uint64_t)(inlen * 8) << 48;
  s.x[1] = K0;
  s.x[2] = K1;
  s.x[3] = 0;
  s.x[4] = 0;
  printstate("initial value", &s);
```

Cryptographic competitions

An illustrated history of Ascon

Real-world challenges

Lessons from usable security

Back to the future of PBC

Why Cryptosystems Fail

Ross Anderson
University Computer Laboratory
Pembroke Street, Cambridge CB2 3QG
Email: rja14@cl.cam.ac.uk

Abstract

Designers of cryptographic systems are at a disadvantage to most other engineers, in that information on how their systems fail is hard to get: their major users have traditionally been government agencies, which are very secretive about their mistakes.

In this article, we present the results of a survey of the failure modes of retail banking systems, which constitute the next largest application of cryptology. It turns out that the threat model commonly used by cryptosystem designers was wrong: most frauds were not caused by cryptanalysis or other technical attacks, but by implementation errors and management failures. This suggests that a paradigm shift is overdue in computer security; we look at some of the alternatives, and see some signs that this shift may be getting under way.

quiries are conducted by experts from organisations with a wide range of interests - the carrier, the insurer, the manufacturer, the airline pilots' union, and the local aviation authority. Their findings are examined by journalists and politicians, discussed in pilots' messes, and passed on by flying instructors.

In short, the flying community has a strong and institutionalised learning mechanism. This is perhaps the main reason why, despite the inherent hazards of flying in large aircraft, which are maintained and piloted by fallible human beings, at hundreds of miles an hour through congested airspace, in bad weather and at night, the risk of being killed on an air journey is only about one in a million.

In the crypto community, on the other hand, there is no such learning mechanism. The history of the subject ([K1], [W1]) shows the same mistakes being made over and over again; in particular, poor management of codebooks

VII-26-X

A HISTORY OF U.S. COMMUNICATIONS SECURITY (U)
(The David G. Boak Lectures)

HANDLING INSTRUCTIONS

1. This publication consists of covers and numbered pages 1 to 101 inclusive. Verify presence of each page upon receipt.
2. Formal authorization for access to ~~SECRET~~ material is required for personnel to have access to this publication.
3. This publication will not be released outside government channels without approval of the Director, National Security Agency.
4. Extracts from this publication may be made for classroom or individual instruction purposes only. Such extracts will be classified ~~SECRET~~ NOFORN and accounted for locally until destroyed.
5. This publication will not be carried in aircraft for use therein.

NATIONAL SECURITY INFORMATION
Unauthorized Disclosure Subject to Criminal Sanctions

NATIONAL SECURITY AGENCY
FORT GEORGE G. MEADE, MARYLAND 20755

Revised July 1973

Classified by Director, NSA, pursuant to NSA Manual 123-2.
Exempt from General Declassification Schedule
of Executive Order 11652 Exempt Category 2.
Declassification date cannot be determined.

~~SECRET~~

ORIGINAL 1
Reverse (Page 2) Blank

DECLASSIFIED UNDER AUTHORITY OF THE
INTERAGENCY SECURITY CLASSIFICATION APPEALS PANEL,
E.O. 13526, SECTION 5.3(b)(3)

ISCAP APPEAL NO. 2009-049, document no. 1
DECLASSIFICATION DATE: October 14, 2015

~~SECRET~~

A HISTORY
OF
U.S. COMMUNICATIONS SECURITY (U)

THE DAVID G. BOAK LECTURES

VOLUME II

NATIONAL SECURITY AGENCY
FORT GEORGE G. MEADE, MARYLAND 20755

The information contained in this publication will not be disclosed to foreign nationals or their representatives without express approval of the DIRECTOR, NATIONAL SECURITY AGENCY. Approval shall refer specifically to this publication or to specific information contained herein.

JULY 1981

CLASSIFIED BY NSA/CSSM 123-2
REVIEW ON 1 JULY 2001

DECLASSIFIED UNDER AUTHORITY OF THE
INTERAGENCY SECURITY CLASSIFICATION APPEALS PANEL,
E.O. 13526, SECTION 5.3(b)(3)

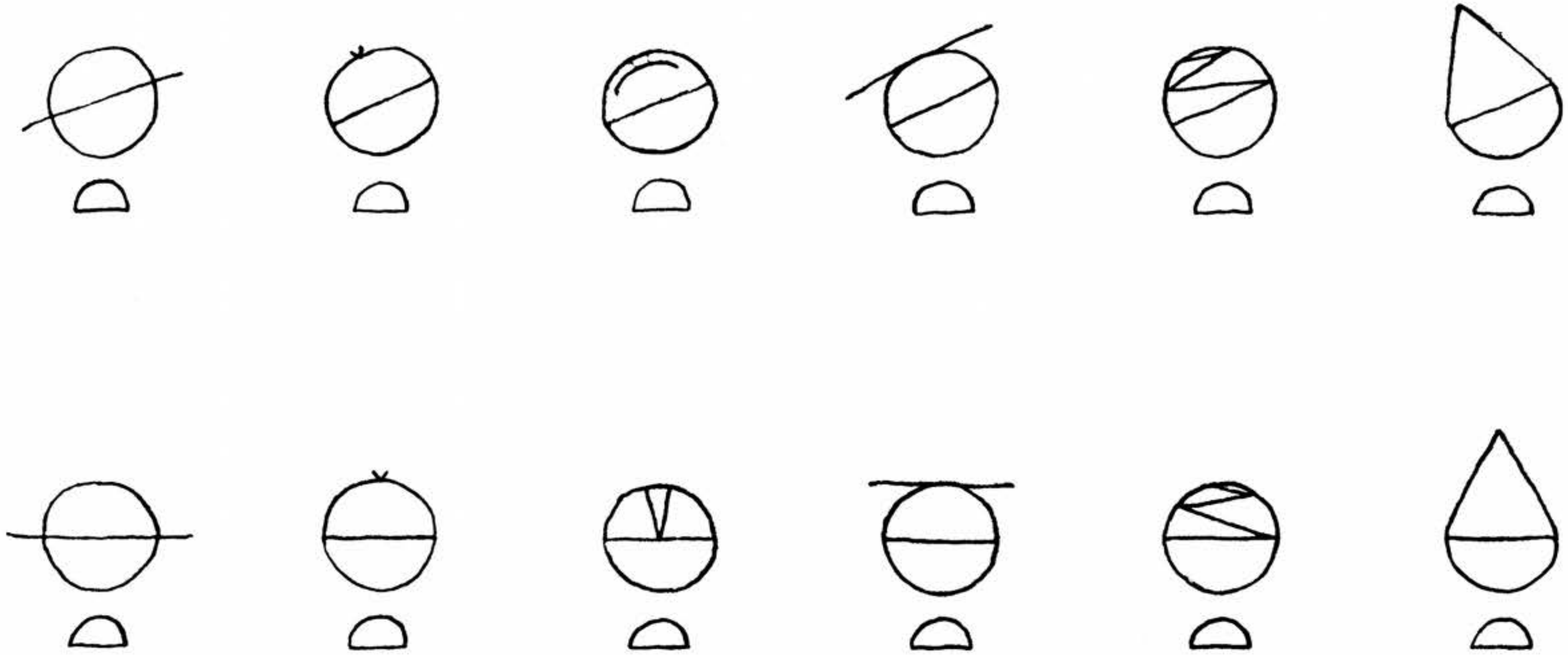
ISCAP APPEAL NO. 2009-049, document no. 2
DECLASSIFICATION DATE: October 14, 2015

NOT RELEASABLE TO FOREIGN NATIONALS

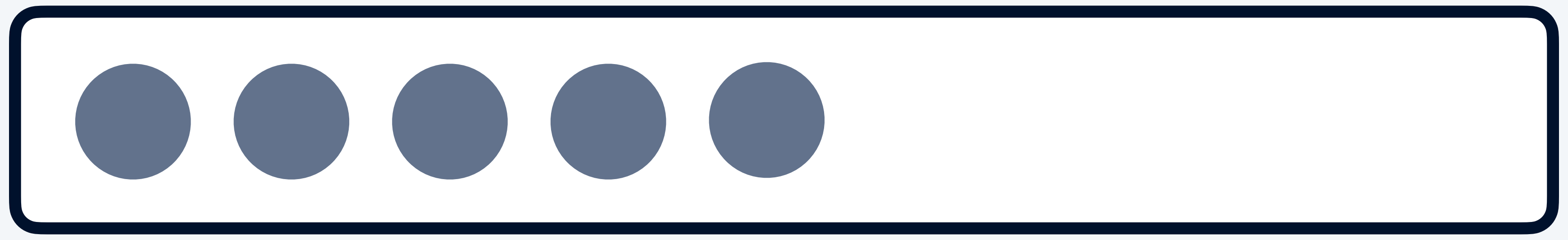
~~SECRET~~
HANDLE VIA COMINT CHANNELS ONLY

ORIGINAL
(Reverse Blank)

[redacted]

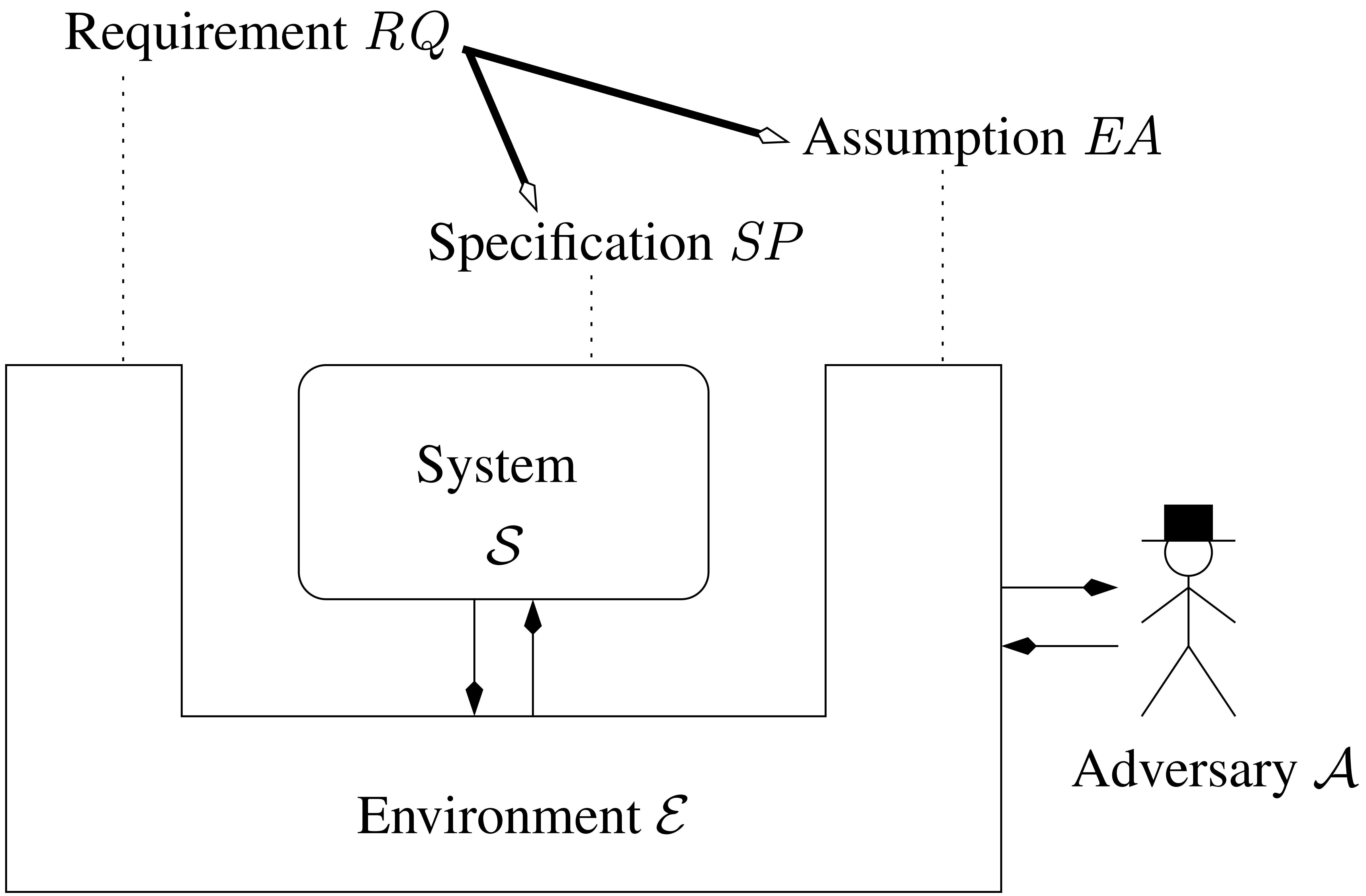


Scott Bluerock, 2006



$$\mathbf{Security(X)} > \mathbf{Security(\bar{X})}$$

$$\mathbf{Outcome(X|ABCD)} > \mathbf{Outcome(\bar{X}|ABCD)}$$



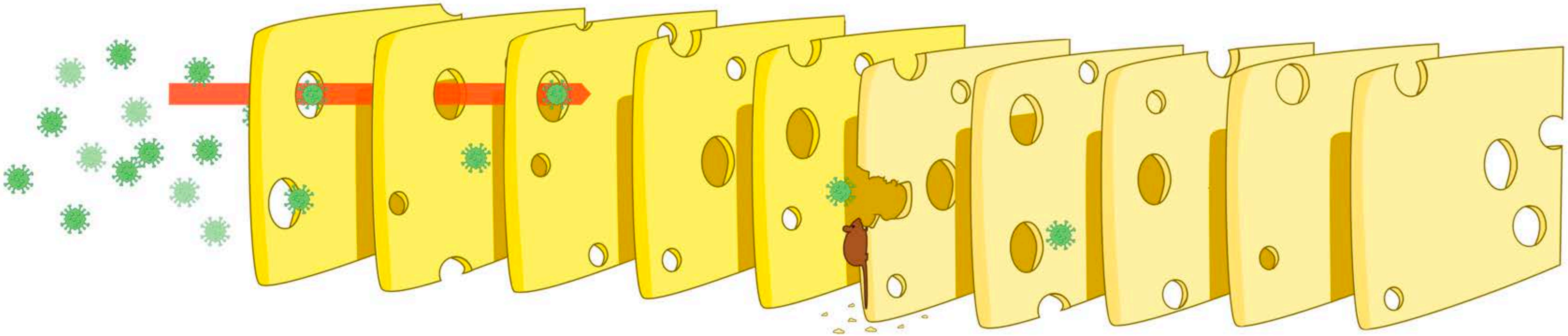
Dashti & Basin, 2016


Postbus dec.rap:
Valideer:
Data vervang:
OpmPriklijst:
Klinische gegevens:
Materiaal selectie:
Afwijk. BTI :
Wijzig trial:
IJsselland
Exit
LAB
Geef patient identificatienummer
Lokaal intranet | Beveiligde modus: uitgeschakeld
10:21
30-10-2015
ThinkVision
bij noodgevallen
112
lenovo
lenovo

KC Green, 2013


H
C. Ross Greening

Tom Nardi, 2019



Ian Mackay, 2020

MHST Educators, 2019

INTERFACE

rsif.royalsocietypublishing.org

Research

Cite this article: Thimbleby H, Oladimeji P, Cairns P. 2015 Unreliable numbers: error and harm induced by bad design can be reduced by better design. *J. R. Soc. Interface* **12**: 20150685.
http://dx.doi.org/10.1098/rsif.2015.0685

Received: 31 July 2015
Accepted: 17 August 2015

Subject Areas:
medical physics

Keywords:
number entry, human error, dependable systems, evaluating user interfaces

Author for correspondence:
Harold Thimbleby
e-mail: harold@thimbleby.net

Unreliable numbers: error and harm induced by bad design can be reduced by better design

Harold Thimbleby[1], Patrick Oladimeji[1] and Paul Cairns[2]

[1]College of Science, Swansea University, Swansea SA2 8PP, UK
[2]Department of Computer Science, University of York, York YO10 5DD, UK

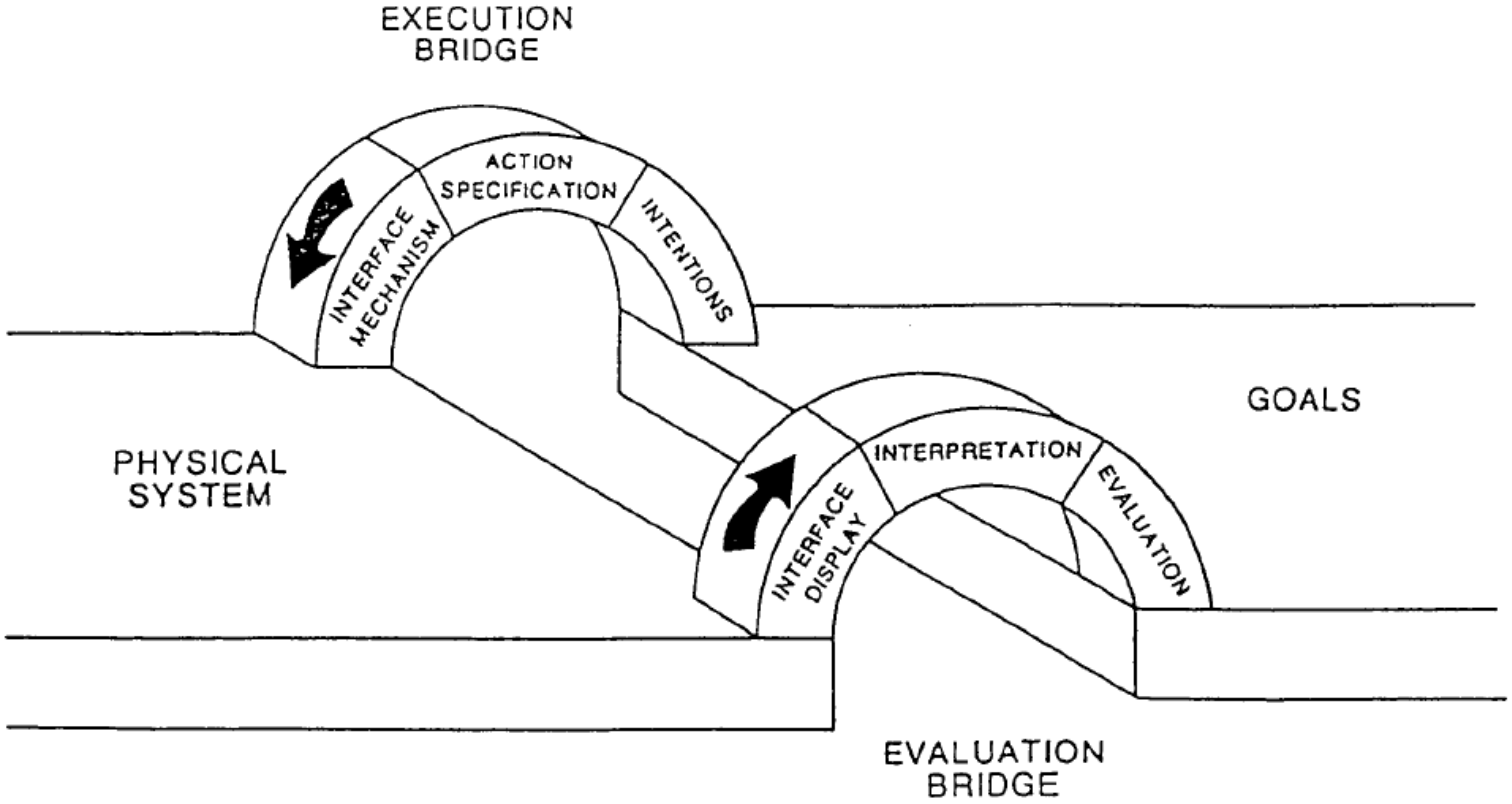
Number entry is a ubiquitous activity and is often performed in safety- and mission-critical procedures, such as healthcare, science, finance, aviation and in many other areas. We show that Monte Carlo methods can quickly and easily compare the reliability of different number entry systems. A surprising finding is that many common, widely used systems are defective, and induce unnecessary human error. We show that Monte Carlo methods enable designers to explore the implications of normal and unexpected operator behaviour, and to design systems to be more resilient to use error. We demonstrate novel designs with improved resilience, implying that the common problems identified and the errors they induce are avoidable.

> Science is a way of trying not to fool yourself. The first principle is that you must not fool yourself, and you are the easiest person to fool.
>
> —Richard P. Feynman [1, ch. 4]

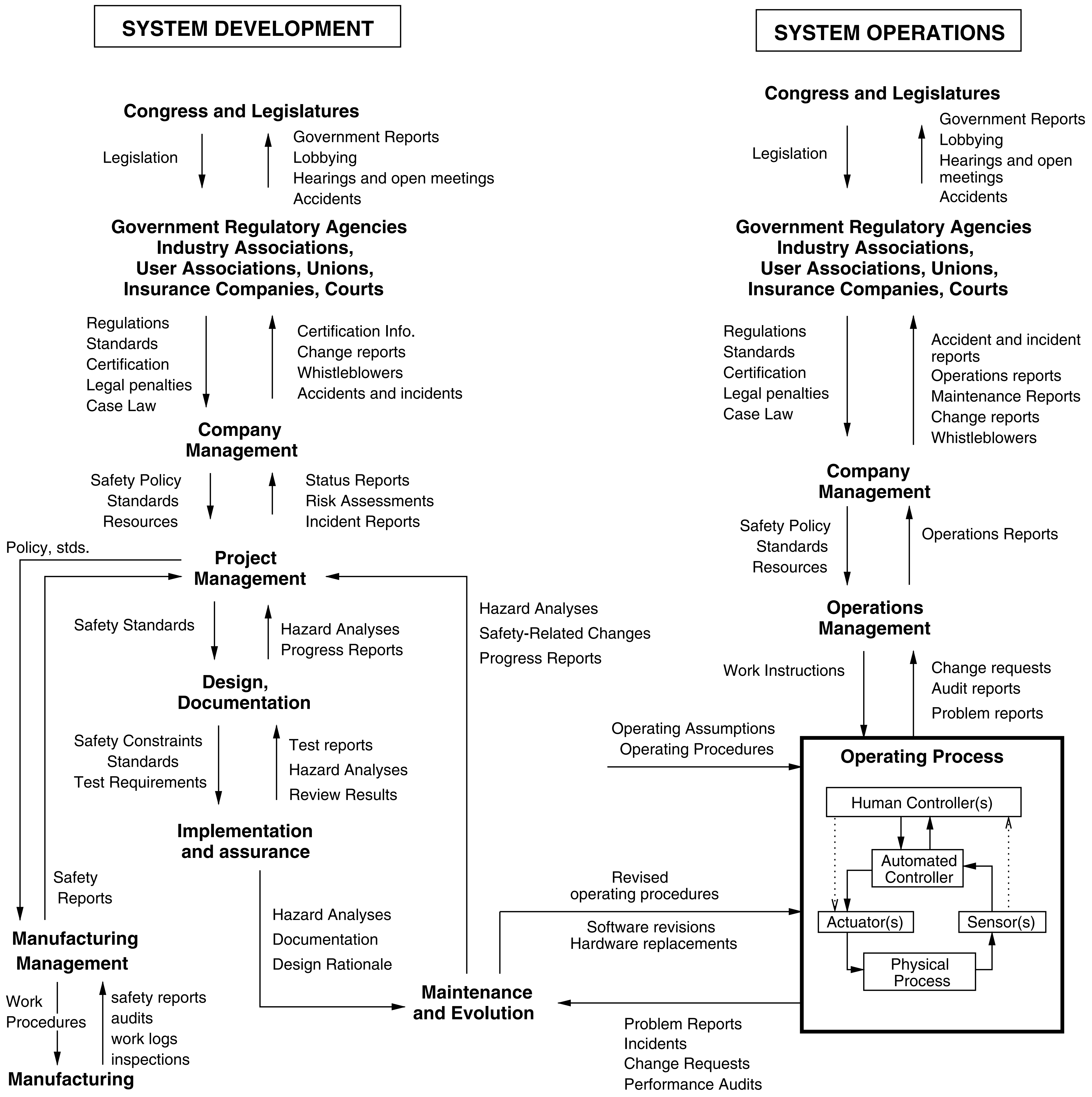
1. Introduction

Number entry is often performed as a 'simple' subtask within a bigger task. For instance, using a calculator typically requires entering a series of numbers and operators. Unnoticed errors while entering the numbers would result in an error in the calculation. To the user who needs to use a calculator and therefore has no precise expectation of the result, this error is likely to go undetected and escalate higher up into the user's workflow or subsequent tasks.

As users of interactive systems, we have little idea how much our unnoticed errors introduce inaccuracy or other problems. Our laboratory work [2] suggests about 3.5% of numbers we enter (on conventional numeric keyboards)



Don Norman, 1986



Nancy Leveson, 2012

Cryptographic competitions

An illustrated history of Ascon

Real-world challenges

Lessons from usable security

Back to the future of PBC

Challenges in Authenticated Encryption

Editor
Daniel J. Bernstein

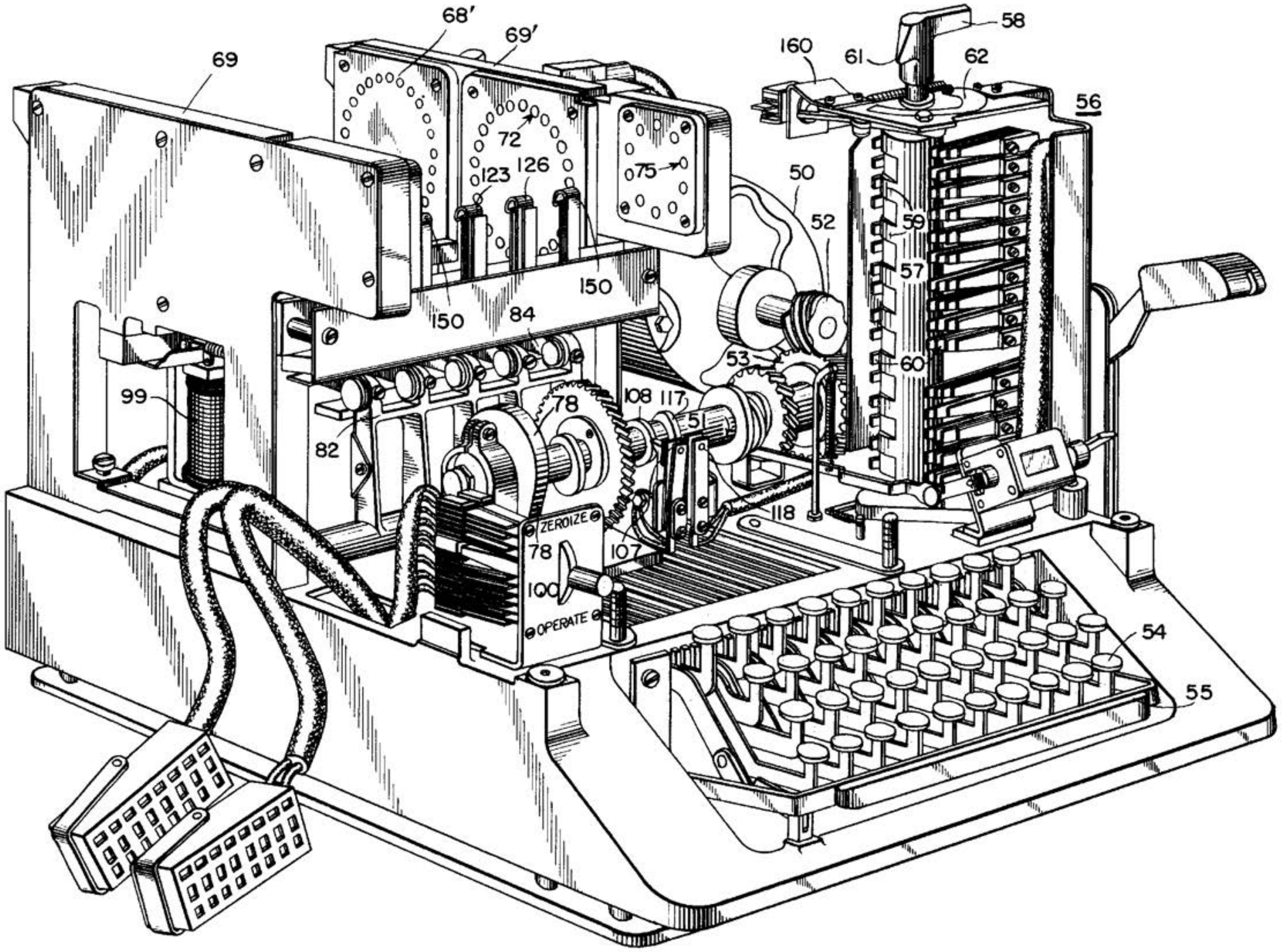
Contributors (alphabetical order; affiliations included for identification only)
Jean-Philippe Aumasson (Kudelski Security, Switzerland)
Steve Babbage (Vodafone, UK)
Daniel J. Bernstein (University of Illinois at Chicago, USA; Technische Universiteit Eindhoven, Netherlands)
Carlos Cid (Royal Holloway, University of London, UK)
Joan Daemen (STMicroelectronics, Belgium; Radboud Universiteit, Netherlands)
Orr Dunkelman (University of Haifa, Israel)
Kris Gaj (George Mason University, USA)
Shay Gueron (University of Haifa, Israel; Intel, Israel)
Pascal Junod (HEIG-VD, Switzerland)
Adam Langley (Google, USA)
David McGrew (Cisco, USA)
Kenny Paterson (Royal Holloway, University of London, UK)
Bart Preneel (KU Leuven, Belgium)
Christian Rechberger (Danmarks Tekniske Universitet, Denmark)
Vincent Rijmen (KU Leuven, Belgium)
Matt Robshaw (Impinj, USA)
Palash Sarkar (Indian Statistical Institute, Kolkata, India)
Patrick Schaumont (Virginia Tech, USA)
Adi Shamir (Weizmann Institute, Israel)
Ingrid Verbauwhede (KU Leuven, Belgium)

17. July 2015 (workshop) + 1. March 2017 (white paper)

Revision 1.05

The work described in this report has in part been supported by the Commission of the European Communities through the H2020-ICT program under contract H2020-ICT-2014 no. 645421.

Contents



Safford & Seiler, 1944

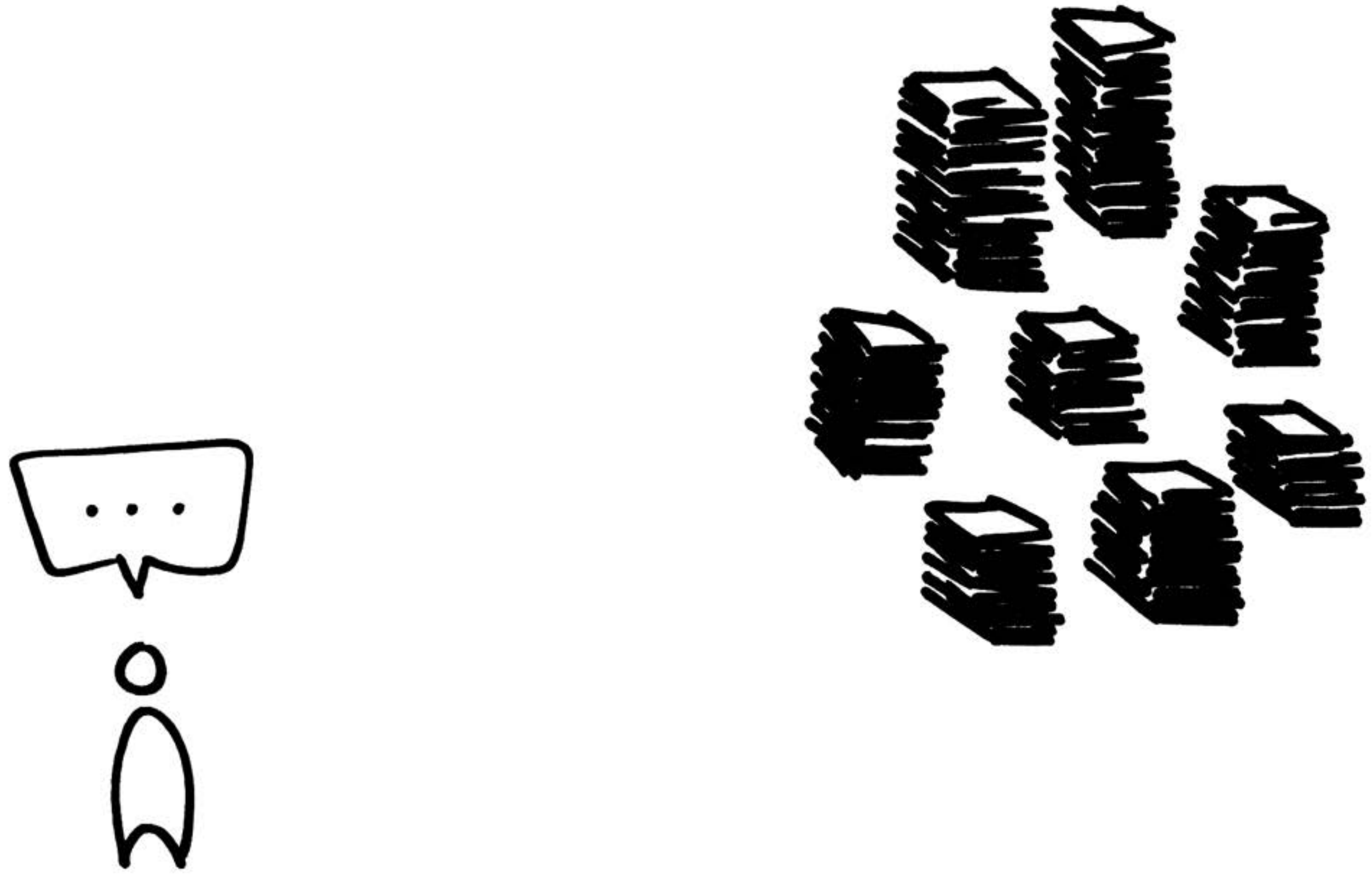
Global Acceptability: While the statutory basis for NIST's work in cryptography is the need for protection of non-national security federal information systems, NIST standards are the foundation of many information technology products and services that are developed by U.S. suppliers and sold globally. NIST recognizes the role of its cryptographic standards in assuring the competitiveness of U.S. industry in delivering these products and services, and is committed to ensuring that its standards and guidelines are accepted internationally.

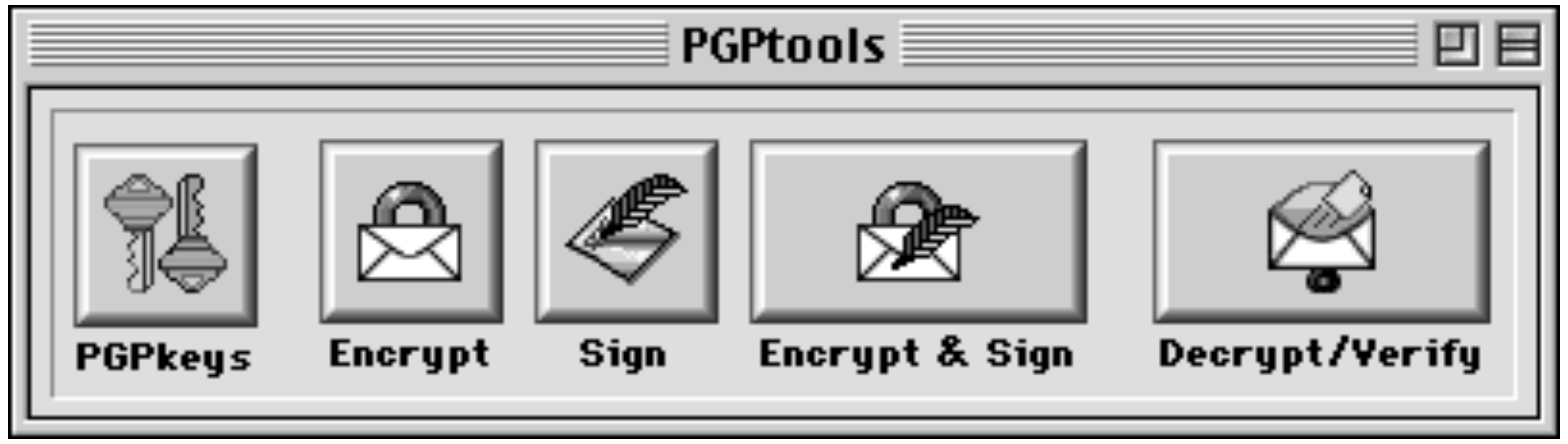
Usability: NIST aims to develop cryptographic standards and guidelines that help implementers create secure and usable systems for their customers that support business needs and workflows, and can be readily integrated with existing and future schemes and systems. Cryptographic standards and guidelines should be chosen to minimize the demands on users and implementers as well as the adverse consequences of human mistakes and equipment failures.

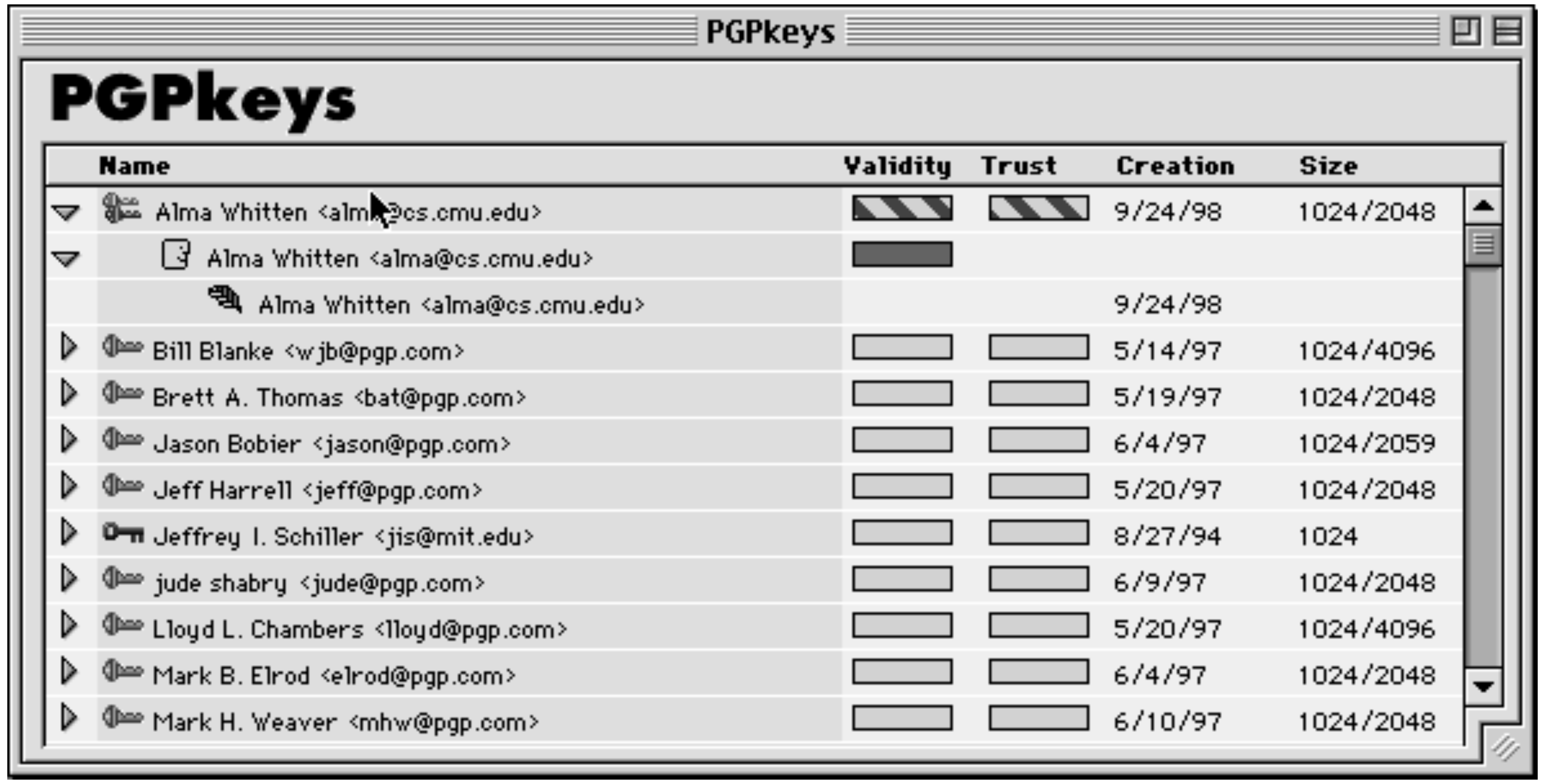
Continuous Improvement: As cryptographic algorithms are developed, and for the duration of their use, the cryptographic community is encouraged to identify weaknesses, vulnerabilities, or other deficiencies in the algorithms specified in NIST publications. When serious problems are identified, NIST engages with the broader cryptographic community to address them. NIST conducts research in order to stay current, to enable new cryptographic advances that may affect the suitability of standards and guidelines, and so that NIST and others can take advantage of those advances to strengthen standards and guidelines.

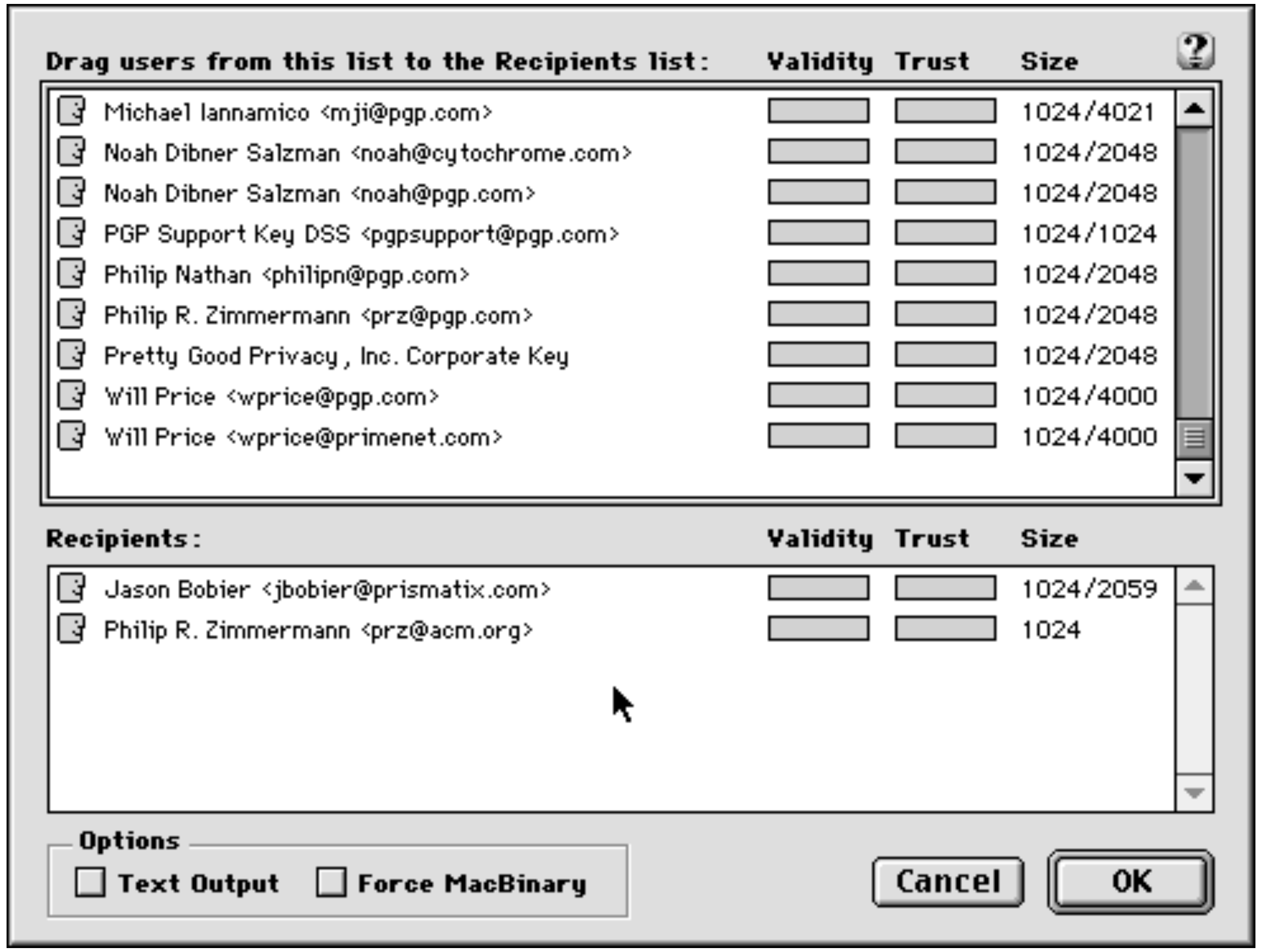
NIST, 2016

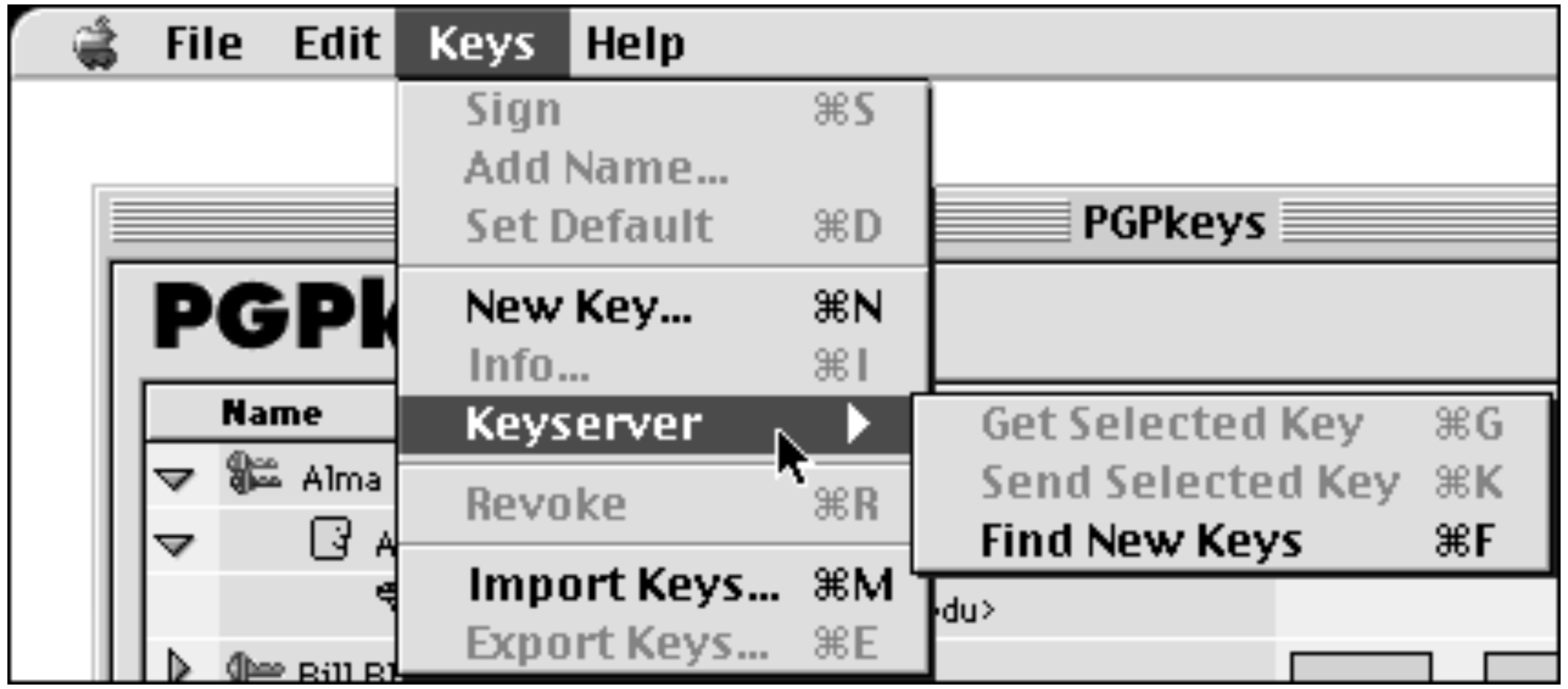
Innovation and Intellectual Property (IP): While developing its cryptographic standards and



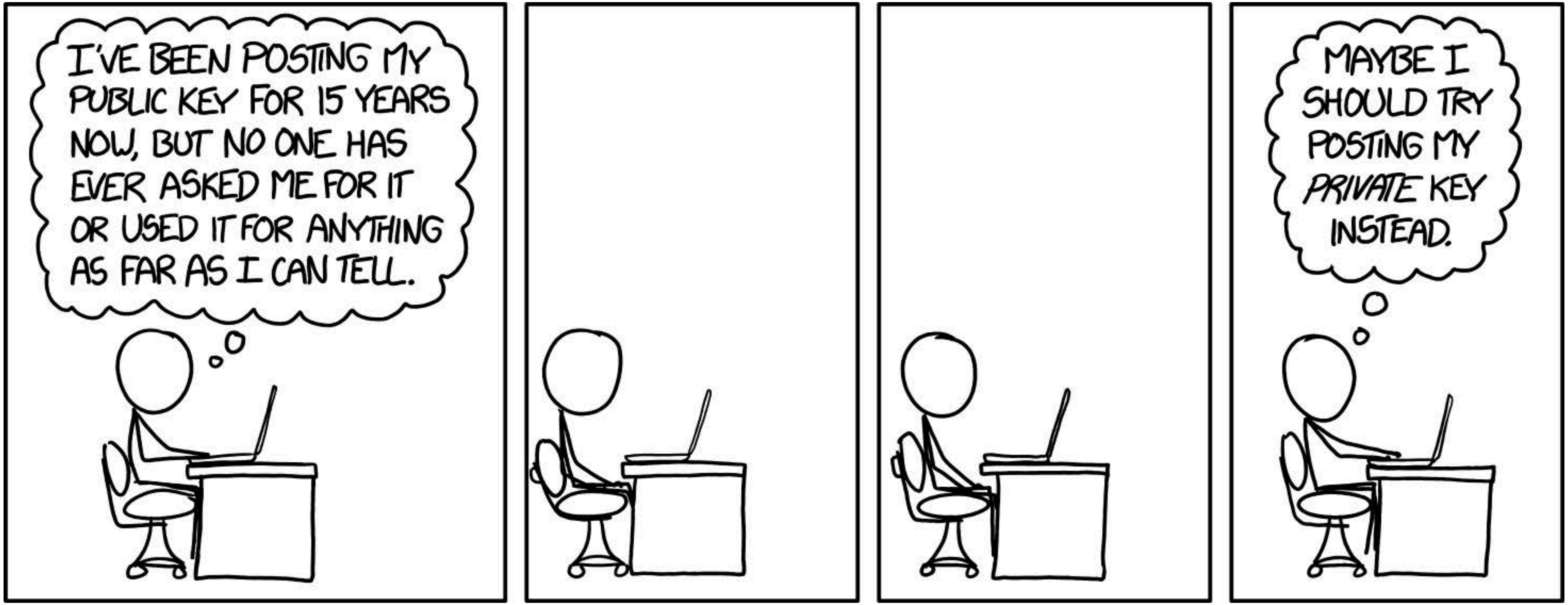




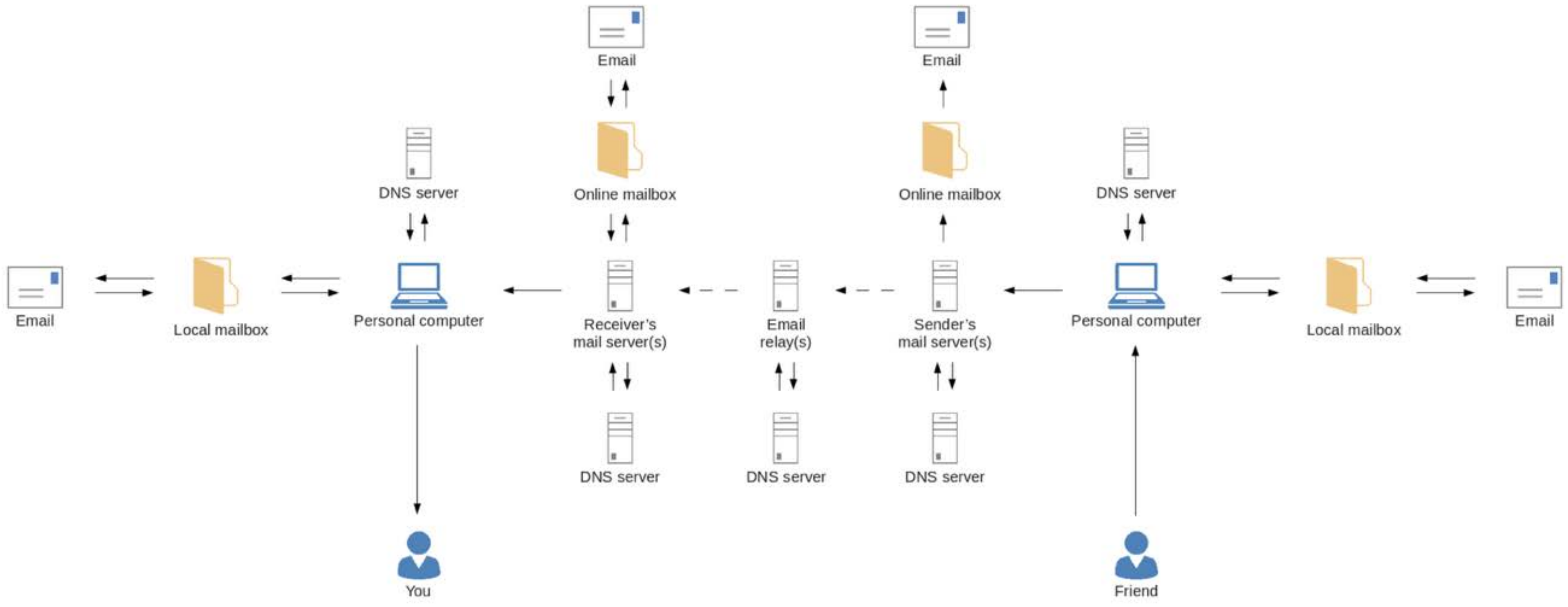




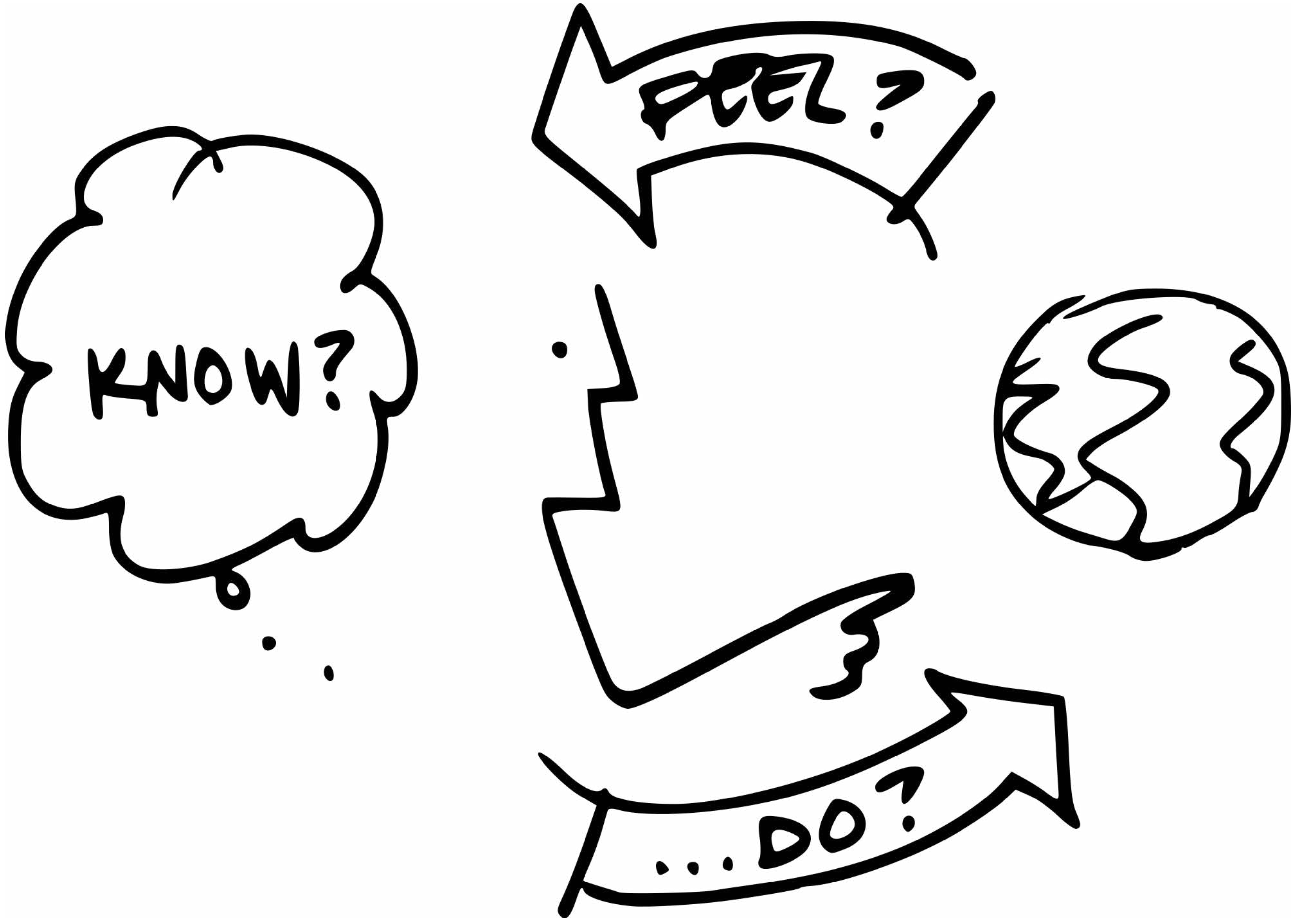
Whitten & Tygar, 1999


I'VE BEEN POSTING MY PUBLIC KEY FOR 15 YEARS NOW, BUT NO ONE HAS EVER ASKED ME FOR IT OR USED IT FOR ANYTHING AS FAR AS I CAN TELL.
MAYBE I SHOULD TRY POSTING MY *PRIVATE* KEY INSTEAD.

xkcd, 2015



Renaud et al., 2014



Bill Verplank, 2000

ascon 0.0.9

```
pip install ascon
```

Latest version

Released: Mar 24, 2023

Lightweight authenticated encryption and hashing

Project description

Python implementation of Ascon

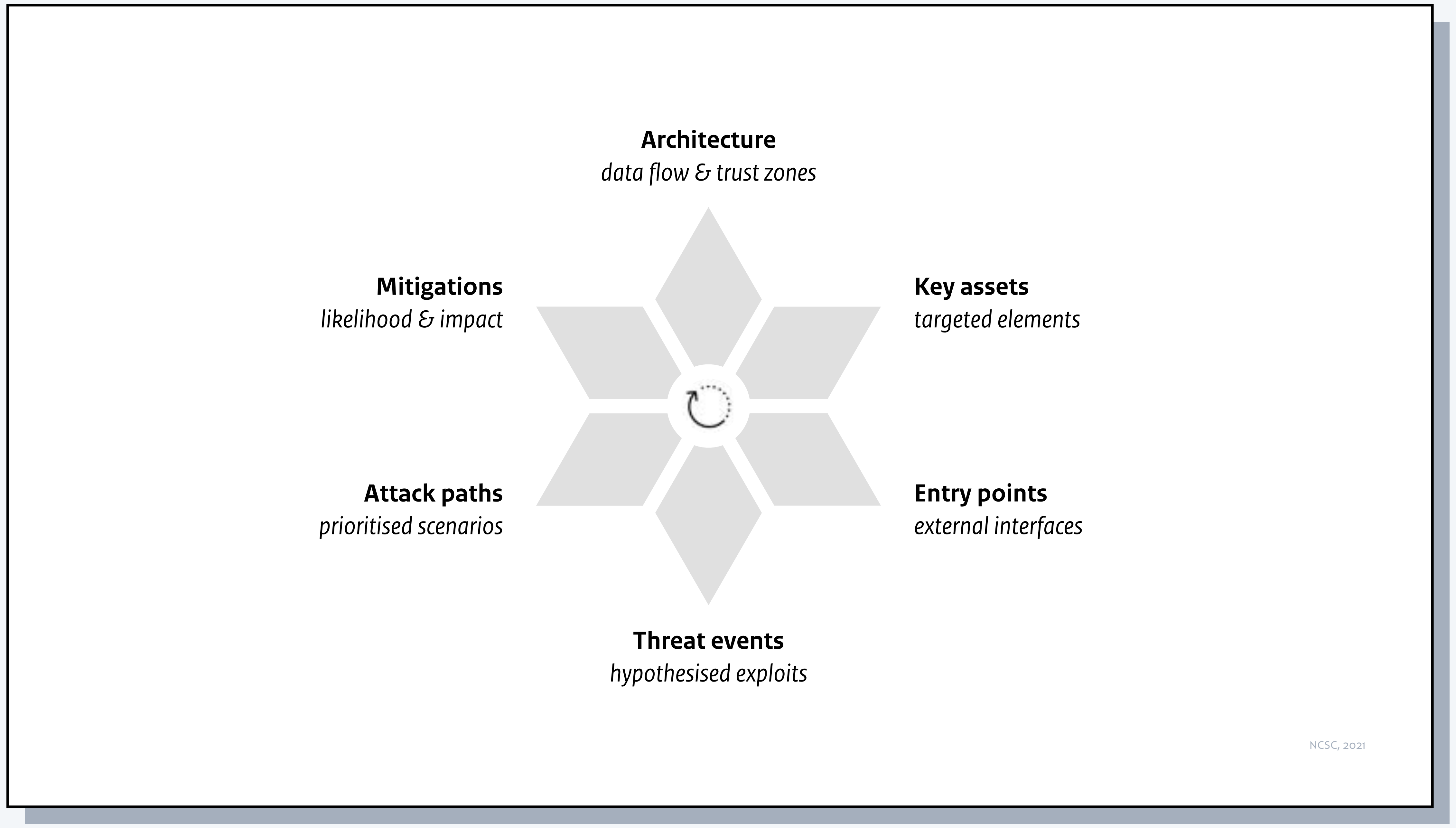
This is a Python3 implementation of Ascon v1.2, an authenticated cipher and hash function.

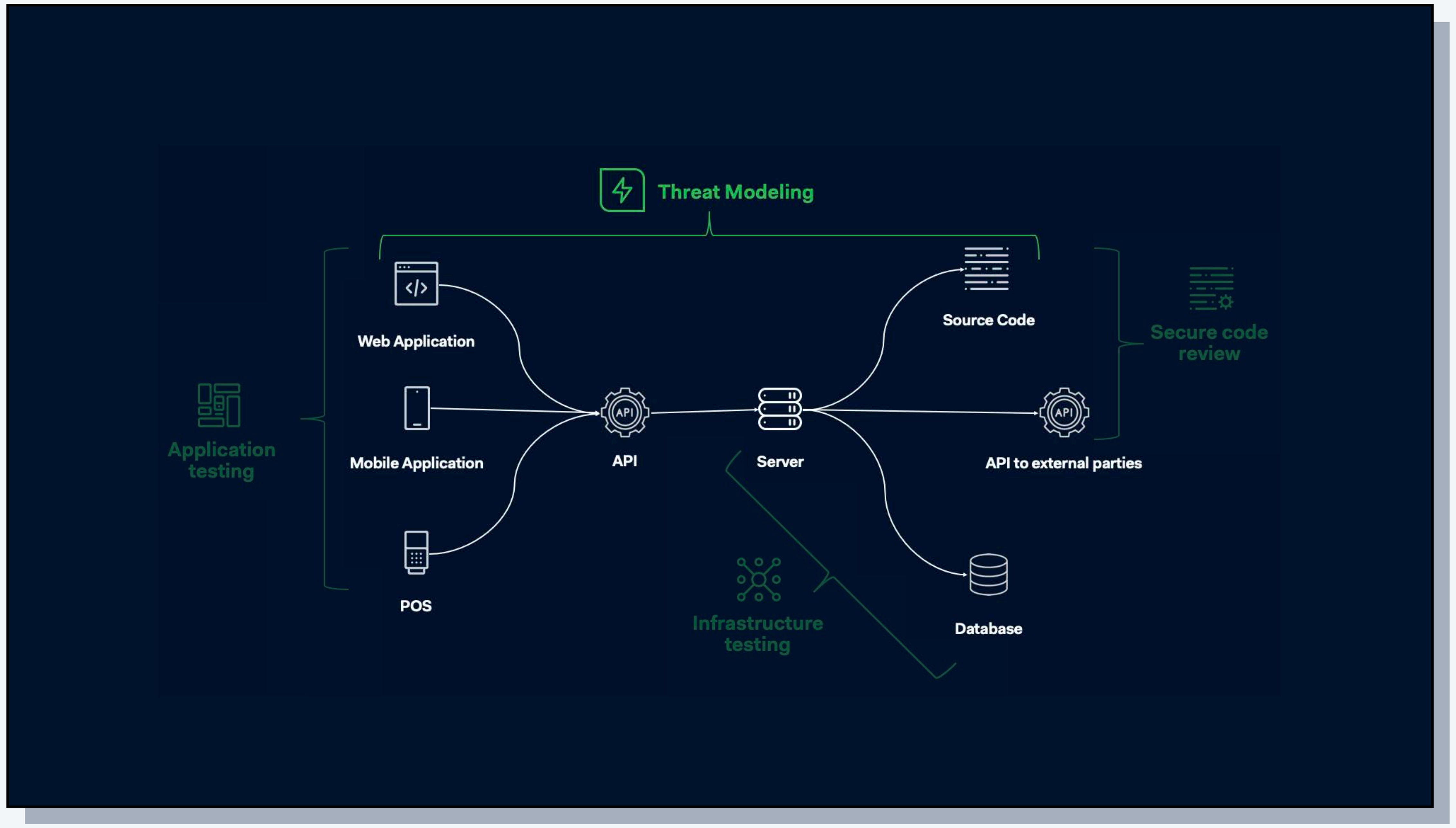
https://github.com/meichlseder/pyascon

Ascon

Ascon is a family of authenticated encryption (AEAD) and hashing algorithms designed to be lightweight and easy to implement, even with added countermeasures against side-channel attacks. It was designed by a team of cryptographers from Graz University of Technology, Infineon Technologies, and Radboud University: Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer.

Ascon has been selected as the standard for lightweight cryptography in the NIST Lightweight Cryptography competition (2019–2023) and as the primary choice for lightweight authenticated encryption in the final portfolio of the CAESAR competition (2014–2019).


Architecture
data flow & trust zones
Mitigations
likelihood & impact
Key assets
targeted elements
Attack paths
prioritised scenarios
Entry points
external interfaces
Threat events
hypothesised exploits
NCSC, 2021


Threat Modeling
Web Application
Source Code
Secure code review
Application testing
Mobile Application
API
Server
API to external parties
POS
Infrastructure testing
Database

Architectural diagrams
with security sauce

Processes
Where data will change from one form to another.

Data flows
Represents data moving from one part of the system to elsewhere.

Data stores
Indicates data at rest, i.e. a place for longer storage.

Terminators
Also called actors or external entities. These are the limits of analysis.

Trust zones
Can be drawn as trust boundaries, i.e. dotted lines between elements.

Confidentiality	Information disclosure
Integrity	Tampering
Availability	Denial of service
Authentication	Spoofing
Authorisation	Elevation of privilege
Accountability	Repudiation

6.3 SR-2: Threat model

6.3.1 Requirement

A process shall be employed to ensure that all products shall have a threat model specific to the current development scope of the product with the following characteristics (where applicable):

a) correct flow of categorized information throughout the system;

b) trust boundaries;

c) processes;

d) data stores;

e) interacting external entities;

f) internal and external communication protocols implemented in the product;

g) externally accessible physical ports including debug ports;

h) circuit board connections such as Joint Test Action Group (JTAG) connections or debug headers which might be used to attack the hardware;

i) potential attack vectors including attacks on the hardware, if applicable;

j) potential threats and their severity as defined by a vulnerability scoring system (for example, CVSS);

k) mitigations and/or dispositions for each threat;

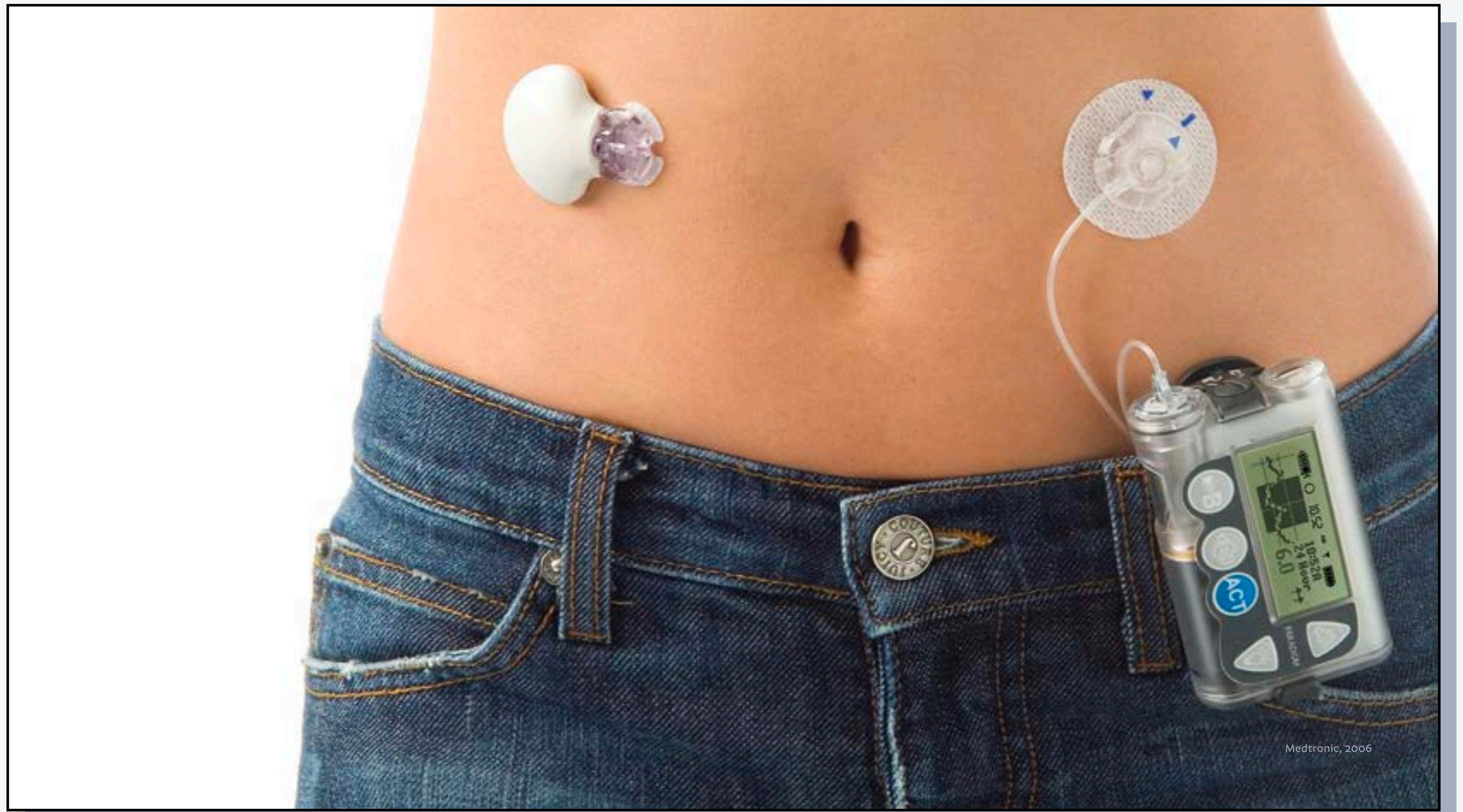
l) security-related issues identified; and

m) external dependencies in the form of drivers or third-party applications (code that is not developed by the supplier) that are linked into the application.

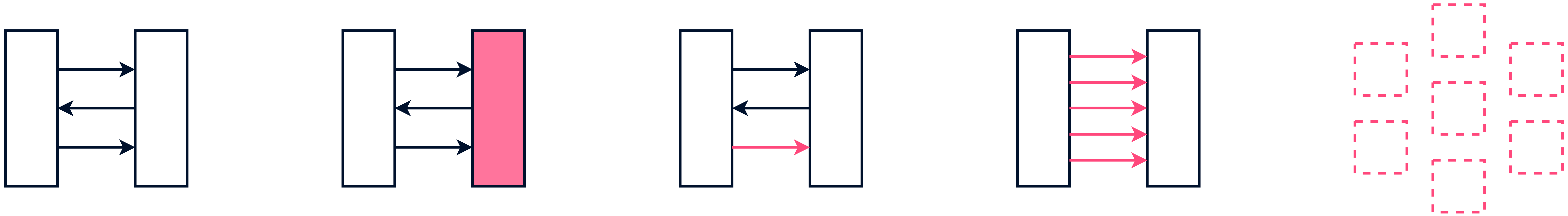
The threat model shall be reviewed and verified by the development team to ensure that it is correct and understood.

The threat model shall be reviewed periodically (at least once a year) for released products and updated if required in response to the emergence of new threats to the product even if the design does not change.

Any issues identified in the threat model shall be addressed as defined in 10.4 and 10.5.



- Components
- Data flows
- Crown jewels
- Trust zones
- Assumptions
- Threats (STRIDE)
- Prioritisation
- Countermeasures
- Security testing
- Follow-up



Silent 'pair programming'

— Don't want to break the flow
— Switch every five minutes
— Apply the refinement approach

10 min. Outline the program's structure as comments
What message(s) will you be sending/receiving?
Which algorithm(s) will you be using for this?

10 min. Write pseudocode to make your ideas tangible

20 min. Translate your pseudocode into Python code

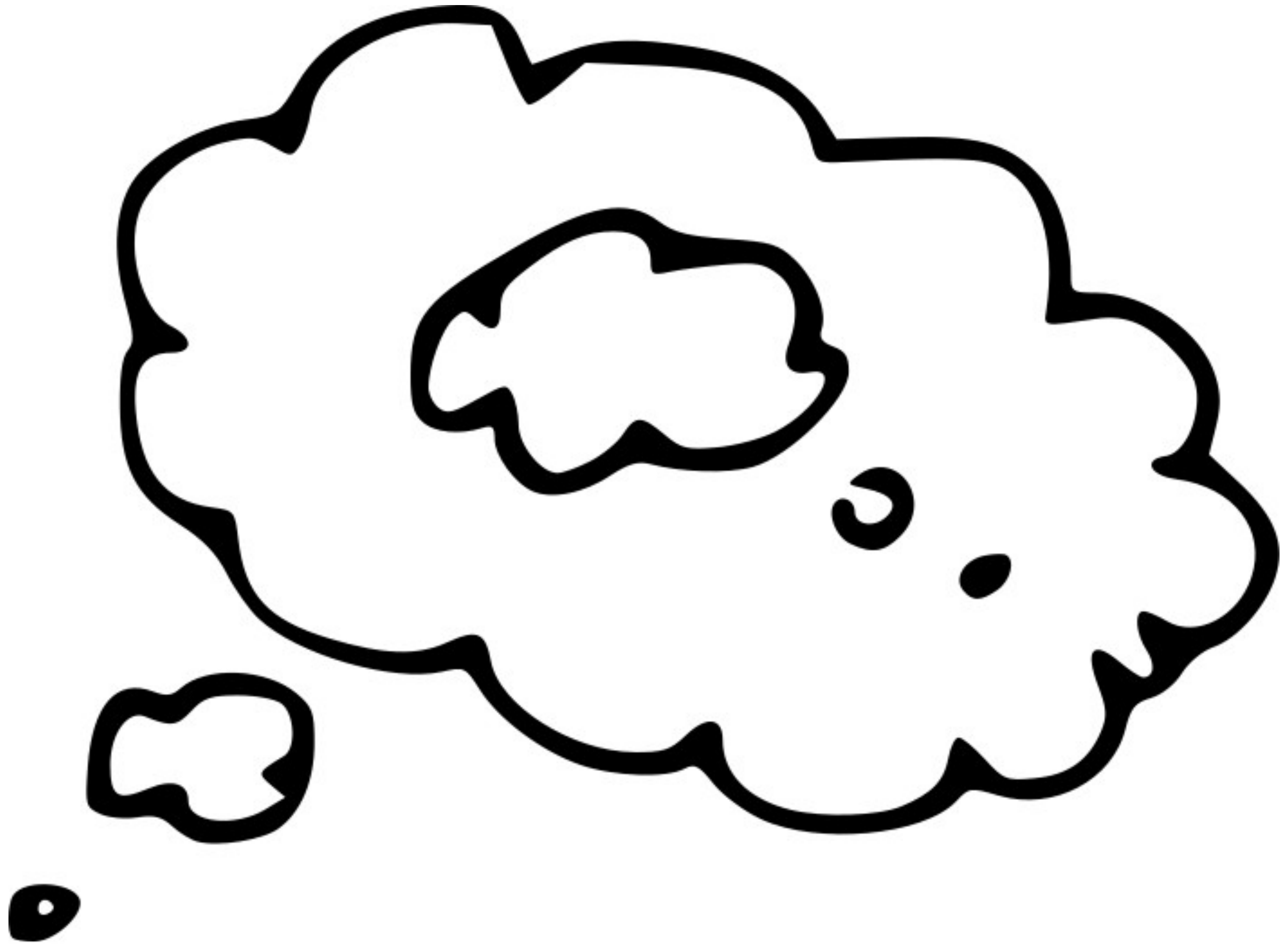
```
https://pypi.org/p/ascon

$ pip install ascon

>>> import ascon
>>> ascon.[tab][tab]
>>> data = b"..."
>>> print(data.hex())
```

Mail your **commented code** to
ascon@arnepadmos.com

Phase 1 – Comments
Alignment of flows and our threat model

Phase 2 – Pseudocode
Match of structure to messages and threats

Phase 3 – Source code
Compare comments to the functions used



Exploratory initial qualitative observations:

- Zero, one, or just a couple of parameters passed
- Wrapper functions taking a message as input
- Hardcoded or empty nonce/key, e.g. in wrapper
- Parameters to library appearing out of thin air
- No key diversification, error handling, etc.

```python
#importing ascon
#create a string that contains the byte string
#sending encrypted data to the sensor
#decrypting the data

import ascon
def encrypt():
    key = b"SECRETSAREHIDDEN"
    message = b"hALLO DIT IS MIJN MESSAGE MET DE VOLGENDE WAARDE: "
    nonce = bytes(16)
    associateddata = b"RELATEDDATA"

    x = ascon.encrypt(key, nonce, associateddata, message, variant="Ascon-80pq")
    return x

#decrypting the data
def decrypt():
    key = b"SECRETSAREHIDDEN"
    nonce = bytes(16)
    associateddata = b"RELATEDDATA"
    y = ascon.decrypt(key, nonce, associateddata, x, variant="Ascon-80pq")
    return y
```

```python
import ascon                                    # Import the ASCON module

ascon = ascon.ASCON()                           # Create an ASCON object

data = b""                                      # Create an empty byte array

for i in range(0, 100):                         # Loop 100 times
    data += bytes([i])                          # Add the current value of i to the data array

def send_encrypted_message(message):            # Define a function to send an encrypted message
    ascon.send(ascon.encrypt(message))          # Encrypt the message and send it

def receive_encrypted_message():                # Define a function to receive an encrypted message
    return ascon.decrypt(ascon.receive())       # Receive the message and decrypt it

def send_encrypted_ack():                       # Define a function to send an encrypted acknoledgement
    ascon.send(ascon.encrypt(b"\x06"))          # Encrypt the acknoledgement and send it

def receive_encrypted_ack():                    # Define a function to receive an encrypted acknoledgement
    return ascon.decrypt(ascon.receive())       # Receive the acknoledgement and decrypt it

print(data.hex())                               # Print the data in hexadecimal format
```

```python
import ascon

def get_data():
    message = ascon.encrypt('give data')
    sensor = 'XX-XX-XX-XX-XX-XX'

    data = ascon.decrypt(send(message,sensor)) # send message to mac sensor and encrypt + [...]

    if data != NULL:
        # data is present so we send the data back
        message = 'ack'
        return data
    elif data = NULL:
        # if no resonse is given, try again
        get_data()

def processdata():
    data = get_data()
    if data < 4.0:
        ins_pump() # send prompt for pump to pump insulin
    elif data > 7.0:
        alert_message() # alert on screen that glucose is too high
```

```
# PHASE 1

# Sensor:
    # send sugarlevels, authentication, checksum
    # send ack received, authentication, checksum
    # send battery level // if battery is low send alert
    # log battery level // if abnormal send alert
    # log connection // if connection behaviour is abnormal drop connection for 10 min.

# Pump:
    # send ack, authentication, checksum
    # log insulin injection

# PHASE 2

# check authentication by checking authentication message
# check integrity by checking the checksum
# check elevation of privilege by checking the log of the battery

# PHASE 3

import ascon
from time import sleep
from time import perf_counter
```

```python
# There is a sensor and a pump, which is sending data from sensor to pump.
# There will be a acknowledgement from pump to sensor.
# The data will be send in integers.
# Spoofing = act as an pump.
# Tampering = interrupt data.
# Information disclosure = intercept and capture sensor data.
# DOS = battery drainage and send garbage.

#Psuedocode
#[...]

def data_encrypt(key, nonce, associateddata, plaintext, data):

    ascon.encrypt(key, nonce, associateddata, plaintext, variant="Ascon-128")
    data = b"blahblahblah"
    print(data.hex())
    return data.hex

def data_decyrpt(key, nonce, associateddata, plaintext, data):

    ascon.decrypt(key, nonce, associateddata, plaintext, variant="Ascon-128a")
    data = b"blahblahblah"
    print(data.hex())
    return data.hex
```

```python
#Sensor:
#   measure blood
#   create uid for message
#   encrypt message + uid
#   Send ecnrypted message to pump
#   if ack with uid not received in less than 10 seconds, send message again.
#   after ack: uid + 1

#Pump:
#   receive data
#   decrypt data
#   send ack to sensor with uid
#   send insuline
#   if uid is lower than or equal to last_uid, drop package
#   otherwise: send insuline and set last_uid to current uid

uid = 1

def sensor_send():
    last_five = []
    measurement = random.choice([1, 2, 3])
    message = str(uid) + ':' + str(measurement)
    b = message.encode('utf-8')
    message = message.hex()
```

Random ideas for future work:

- Use of ‘AEAD’ and ‘XOF’, not ‘MAC’ or ‘hash’
- Define standard serialisation, e.g. AD | n | C | t
- Appropriate parameter ordering for functions
- Creation of a compatible user-friendly wrapper
- Impact of programming paradigm on output

RC 9265 (#40713) 2/17/82
Computer Science 6 pages

Using the "Thinking-aloud" Method in Cognitive Interface Design

Clayton Lewis

IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

Abstract: "Thinking-aloud" is a method for studying mental processes in which participants are asked to make spoken comments as they work on a task. The method is appropriate for studying the cognitive problems that people have in learning to use a computer system. This note discusses the strengths and weaknesses of the method, and gives some suggestions about its use based on laboratory experience at Yorktown.

IDEO.org, 2015

SCA Evaluation & Benchmarking of Finalists in the NIST Lightweight Cryptography Standardization Process

Jens-Peter Kaps
& Kris Gaj

Cryptographic Engineering Research Group
cryptography.gmu.edu

George Mason University

How might we integrate usability into our process?

How would you like your designs to be evaluated?

Cryptographic competitions

An illustrated history of Ascon

Real-world challenges

Lessons from usable security

Back to the future of PBC

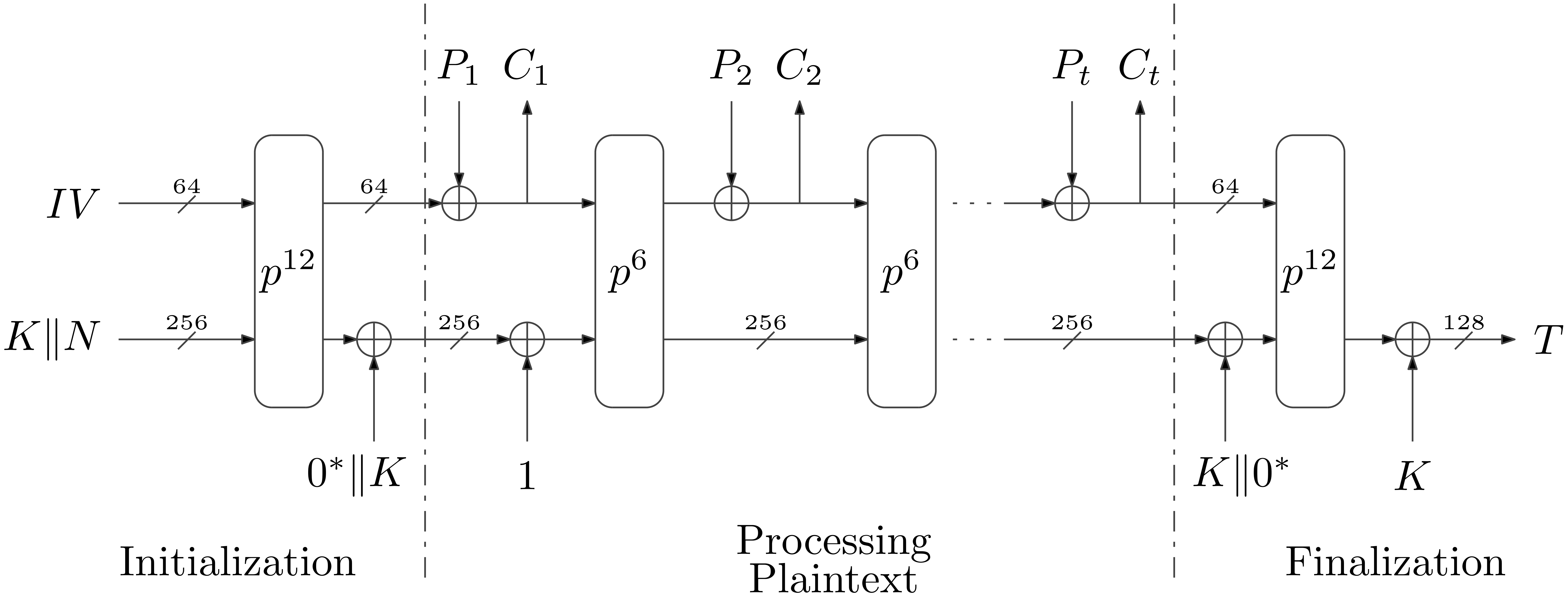
Next Steps

- Publication of the third–round status update
- Sixth Lightweight Cryptography Workshop in June 21-22 2023 (virtual)

  Submission deadline: May 1, 2023

  Aim: to explain the selection process, and to discuss various aspects of lightweight cryptography standardization, such as

  - Which ASCON variants to standardize? All of subset ? XOF instead of hash?
  - Additionally functionality, e.g. dedicated MAC?
  - Support for additional parameter sizes? e.g., larger nonce, shorter tags
- Publication of draft standard (in 2023)



Christoph Dobraunig, 2018

From: hi@arnepadmos.com
Sent: Sunday, October 9, 2022 5:13 PM
To: lightweight-crypto
Cc: lwc-forum@list.nist.gov
Subject: FINALIST OFFICIAL COMMENT: ASCON

Dear NIST,

Let me start by saying that I think Ascon would make a great selection for the NIST LWC standard. I do have several comments:

Ascon parameters -- While the 30 September 2022 status update about Ascon states that the authors 'consider both Ascon-128 and Ascon-128a to be equally well-suited and secure choices', during both the CAESAR and LWC competition, Ascon-128 has always been the primary recommendation in every version of the submitted specifications. I believe that Ascon-128 should remain the primary recommendation, as I think that 'late' changes of key decisions -- such as those made to Romulus -- are undesirable.

Sessions and ratcheting -- In the latest Xoodyak update, the authors emphasise 'that API flexibility is an important asset for a lightweight cryptographic primitive'. Specifically, they note the utility of support for sessions and rolling subkeys. In personal communication, the Ascon team has shared that intermediate tags and ratcheting can be implemented by reusing the MAC as the nonce and by using the non-masked half of the state as the new key. If Ascon is selected, I believe it would be useful to standardise such features in an additional publication (see below).

Feature parity with SHAKE -- One year after SHA-3 was standardised as FIPS 202, an extension defining modes of operation constructed around SHAKE was published as NIST SP 800-185. Key features of these modes are the support for tuples and customisation strings. In addition to support for sessions and ratcheting, Ascon can also benefit from such

Adam Cohn, 2006

From a security point of view, an AEAD algorithm should ensure both the confidentiality of the plaintexts (under adaptive chosen-plaintext attacks) and the integrity of the ciphertexts (under adaptive forgery attempts). AEAD algorithms are expected to maintain security as long as the nonce is unique (not repeated under the same key). Any security loss when the nonce is not unique **shall** be documented, and algorithms that do not lose all security with repeated nonces may advertise this as a feature.

The submitters are allowed to submit a family of AEAD algorithms, where members of the family may vary in external parameters (e.g., key length, nonce length), or in internal parameters (e.g., number of rounds, or state size). The family **shall** include at most 10 members. The following requirements apply to all members of the family.

An AEAD algorithm **shall** not specify key lengths that are smaller than 128 bits. Cryptanalytic attacks on the AEAD algorithm **shall** require at least 2^{112} computations on a classical computer in a single-key setting. If a key size larger than 128 bits is supported, it is recommended that at least one recommended parameter set has a key size of 256 bits, and that its resistance against cryptanalytical attacks is at least 2^{224} computations on a classical computer in a single-key setting.

AEAD algorithms **shall** accept all byte-string inputs that satisfy the input length requirements. Submissions **shall** include justification for any length limits.

The family **shall** include one *primary* member that has a key length of at least 128 bits, a nonce length of at least 96 bits, and a tag length of at least 64 bits. The limits on the input sizes (plaintext, associated data, and the amount of data that can be processed under one key) for this member **shall not** be smaller than 2^{50}-1 bytes.

ks_i	:	The keystream bit generated at the ith step.
$pclen$	:	bit length of the plaintext/ciphertext with $0 \le pclen < 2^{64}$.
m_i	:	one data bit.
P	:	plaintext.
p_i	:	the ith plaintext bit.
S_i	:	state at the beginning of the ith step.
$S_{i,j}$	:	jth bit of state S_i . For ACORN-128, $0 \le j \le 292$.
T	:	authentication tag.
t	:	bit length of the authentication tag with $64 \le t \le 128$.

1.2.3 Functions

Two Boolean functions are used in ACORN: maj and ch.

$$maj(x, y, z) = (x\&y) \oplus (x\&z) \oplus (y\&z) \ ;$$
$$ch(x, y, z) = (x\&y) \oplus ((\sim x)\&z) \ ;$$

1.3 ACORN-128

ACORN-128 uses a 128-bit key and a 128-bit initialization vector. The associated data length and the plaintext length are less than 2^{64} bits. The authentication tag length is less than or equal to 128 bits. We recommend the use of a 128-bit tag.

1.3.1 The state of ACORN-128

The state size of ACORN-128 is 293 bits. There are six LFSRs being concatenated in ACORN-128. The state is shown in Fig.1.1.

I'd rather not add a new, dedicated MAC mode of operation unless it provides an advantage that Ascon-AEAD can not. That advantage should then be clearly stated.

Simon Hoerder, 2023

The discussion of customization strings got me thinking.

When hashing with either ASCON-HASH or ASCON-XOF, the first block is pre-formatted with 8 bytes of IV / domain separation material, including the desired output length in bits. The other 32 bytes are currently set to zero. The permutation is applied and then absorbing begins. This first permutation invocation can be pre-computed if the output length is known at compile time.

This leaves 32 bytes that could be used for algorithm names (e.g. "KMAC" or "KDF" or "TupleHash" or whatever) and/or customisation strings. If the customisation data is too large for a single block, then the permutation can be iterated to absorb the remaining bytes with a domain separation bit set to distinguish customisation data from regular data.

Here is a pseudocode outline of one possible encoding with the algorithm name in the first block and the customisation string absorbed separately:

```
IF len(AlgorithmName) > 32 THEN
   AlgorithmName = ASCON-HASH(AlgorithmName)
ENDIF
FirstBlock = {8-byte IV} || pad-with-zeroes(AlgorithmName, 32)
S = ASCON_p(FirstBlock)
IF len(CustomString) > 0 THEN
   C = pad-to-rate(CustomString || 1 || zeroes)
   absorb C into S in rate-sized chunks, with the domain separation bit XOR'ed with 1 in each chunk
ENDIF
absorb the input data into S
squeeze the output data from S
```

An empty algorithm name and customisation string would be equivalent to the current behaviour.

Cheers,

Rhys.

A given instance, denoted TurboSHAKE[c], takes as input:

- a message M, a byte string of variable length, and
- a domain separation parameter D, a byte with a value in the range [`0x01`, ..., `0x7F`] in hexadecimal.

As a XOF, the output of TurboSHAKE[c] is unlimited, and the user can request as many output bits as desired. It can be used for traditional hashing simply by generating outputs of the desired digest size.

TurboSHAKE produces unrelated outputs on different tuples (c, M, D). For a given capacity, the value D is meant to provide domain separation, that is, for two different values $D_1 \neq D_2$, TurboSHAKE[c]$(\cdot, D_1)$ and TurboSHAKE[c]$(\cdot, D_2)$ act as two independent functions of M. We believe the range of D to be sufficient to cover all use cases.

Users that do not require multiple instances can take as default $D =$ `0x1F`.

Named instances In addition, we define:

- TurboSHAKE128 as TurboSHAKE[$c = 256$], and
- TurboSHAKE256 as TurboSHAKE[$c = 512$].

Procedure To compute TurboSHAKE[c](M, D), proceed as follows. Let $R = 200 - c/8$ be the rate in bytes and f the Keccak-$p[1600, n_r = 12]$ permutation [60].

Bertoni et al., 2023

1. Input preparation

Table 2.3.: Initial values for ISAP instances in hex notation.

ISAP-$\mathcal{P}$-$^{r_H,r_B}_{s_H,s_B,s_E,s_K}$-$k$	IV_A	$1 \parallel k \parallel r_H \parallel r_B \parallel s_H \parallel s_B \parallel s_E \parallel s_K \parallel 0^*$
	IV_{KA}	$2 \parallel k \parallel r_H \parallel r_B \parallel s_H \parallel s_B \parallel s_E \parallel s_K \parallel 0^*$
	IV_{KE}	$3 \parallel k \parallel r_H \parallel r_B \parallel s_H \parallel s_B \parallel s_E \parallel s_K \parallel 0^*$
ISAP-A-128A	IV_A	01 80 4001 0C01060C 00*
	IV_{KA}	02 80 4001 0C01060C 00*
	IV_{KE}	03 80 4001 0C01060C 00*
ISAP-K-128A	IV_A	01 80 9001 10010808 00*
	IV_{KA}	02 80 9001 10010808 00*
	IV_{KE}	03 80 9001 10010808 00*
ISAP-A-128	IV_A	01 80 4001 0C0C0C0C 00*
	IV_{KA}	02 80 4001 0C0C0C0C 00*
	IV_{KE}	03 80 4001 0C0C0C0C 00*
ISAP-K-128	IV_A	01 80 9001 140C0C0C 00*
	IV_{KA}	02 80 9001 140C0C0C 00*
	IV_{KE}	03 80 9001 140C0C0C 00*

2.6. On Hash Functions using ASCON-p or KECCAK-$p[400]$

Since ISAP is based on either ASCON-p or KECCAK-$p[400]$, it lends itself to pairing with already specified hash functions using the same permutations. In the case of ISAP-A-128A and ISAP-A-128, we suggest a pairing with the hash function ASCONHASH specified in the

Dobraunig et al., 2021

As illustrated by BLINKER, Strobe, SHOE, and Cyclist, sponges can be the basis for simple, lightweight two party half-duplex record protocols. **Support for tuples** and customisation strings – e.g. through additional domain separation constants and/or padding rules – can disambiguate directionality, metadata, headers, and protocol types.

From my comments to NIST

Parsing ambiguities in authentication and key establishment protocols

Liqun Chen
Hewlett-Packard Laboratories
Filton Road
Stoke Gifford
Bristol BS34 8QZ, UK
liqun.chen@hp.com

Chris J. Mitchell
Royal Holloway
University of London
Egham
Surrey TW20 0EX, UK
c.mitchell@rhul.ac.uk

30th September 2008

Abstract

A new class of attacks against authentication and authenticated key establishment protocols is described, which we call *parsing ambiguity attacks*. If appropriate precautions are not deployed, these attacks apply to a very wide range of such protocols, including those specified in a number of international standards. Three example attacks are described in detail, and possible generalisations are also outlined. Finally, possible countermeasures are given, as are recommendations for modifications to the relevant standards.

1 Introduction

Over the last four years a number of new attacks have been published on long-established and apparently stable standardised authenticated key establishment protocols. The origin of these protocols can be traced back to the seminal paper of Needham and Schroeder [24], and the protocols concerned had been widely studied and were believed to be secure. Indeed, the first edition of the international standard for key establishment mechanisms using symmetric cryptography, ISO/IEC 11770-2, appeared in 1996 [8], and no problems were identified until 2004.

However, things have changed in recent years, with the publication of a number of attacks (including a range of 'type attacks') on two standardised protocols. The attacked protocols (mechanisms 12 and 13 of ISO/IEC 11770-2) both assume that the two parties who wish to establish a shared secret key already share a secret key with a trusted third party (acting as a key translation centre).

ALPACA: Application Layer Protocol Confusion - Analyzing and Mitigating Cracks in TLS Authentication

Marcus Brinkmann[1], Christian Dresen[2], Robert Merget[1], Damian Poddebniak[2], Jens Müller[1], Juraj Somorovsky[3], Jörg Schwenk[1], and Sebastian Schinzel[2]

[1]Ruhr University Bochum
[2]Münster University of Applied Sciences
[3]Paderborn University

Abstract

TLS is widely used to add confidentiality, authenticity and integrity to application layer protocols such as HTTP, SMTP, IMAP, POP3, and FTP. However, TLS does not bind a TCP connection to the intended application layer protocol. This allows a man-in-the-middle attacker to redirect TLS traffic to a different TLS service endpoint on another IP address and/or port. For example, if subdomains share a wildcard certificate, an attacker can redirect traffic from one subdomain to another, resulting in a valid TLS session. This breaks the authentication of TLS and *cross-protocol attacks* may be possible where the behavior of one service may compromise the security of the other at the application layer.

In this paper, we investigate cross-protocol attacks on TLS in general and conduct a systematic case study on web servers, redirecting HTTPS requests from a victim's web browser to SMTP, IMAP, POP3, and FTP servers. We show that in realistic scenarios, the attacker can extract session cookies and other private user data or execute arbitrary JavaScript in the context of the vulnerable web server, therefore bypassing TLS and web application security.

We evaluate the real-world attack surface of web browsers and widely-deployed email and FTP servers in lab experiments and with internet-wide scans. We find that 1.4M web servers are generally vulnerable to cross-protocol attacks, i.e., TLS application data confusion is possible. Of these, 114k web servers can be attacked using an exploitable application server. Finally, we discuss the effectiveness of TLS extensions such as Application Layer Protocol Negotiation (ALPN) and Server Name Indiciation (SNI) in mitigating these and other cross-protocol attacks.

1 Introduction

TLS. With Transport Layer Security (TLS) [56], confidential and authenticated channels are established between two communication endpoints. In typical end-user protocols, such as HTTP, SMTP, or IMAP, the TLS server authenticates to the client by presenting an X.509 certificate. In this setting, the server is identified by the *Common Name* (CN) field or the *Subject Alternate Name* (SAN) extension in the certificate, which contains one or more hostnames or wildcard patterns (e.g., `*.bank.com`). As part of the certificate validation, the client confirms that the destination of the request matches the CN or SAN of the certificate.

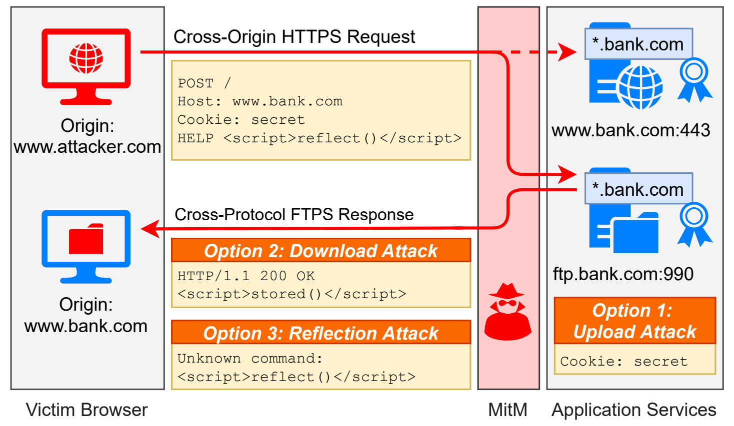


Figure 1: Basic idea behind application layer cross-protocol attacks on HTTPS. A MitM attacker leads the victim to an attacker-controlled website that triggers a cross-origin HTTPS request with a specially crafted FTP payload. The attacker then redirects the request to an FTP server that has a certificate compatible with the web server. The attack either (1) uploads a secret cookie to FTP, or (2) downloads a stored malicious JavaScript file from FTP, or (3) reflects malicious JavaScript contained in the request. In case (2) and (3), the JavaScript code is executed in the context of the targeted web service.

Since TLS does not protect the integrity of the TCP connection itself (i.e., source IP & port, destination IP & port), a man-in-the-middle (MitM) attacker can redirect TLS traffic for the *intended* TLS service endpoint and protocol to another, *substitute* TLS service endpoint and protocol. If the client considers the certificate of the substitute server to be valid for the intended server, for example, if wildcard certificates

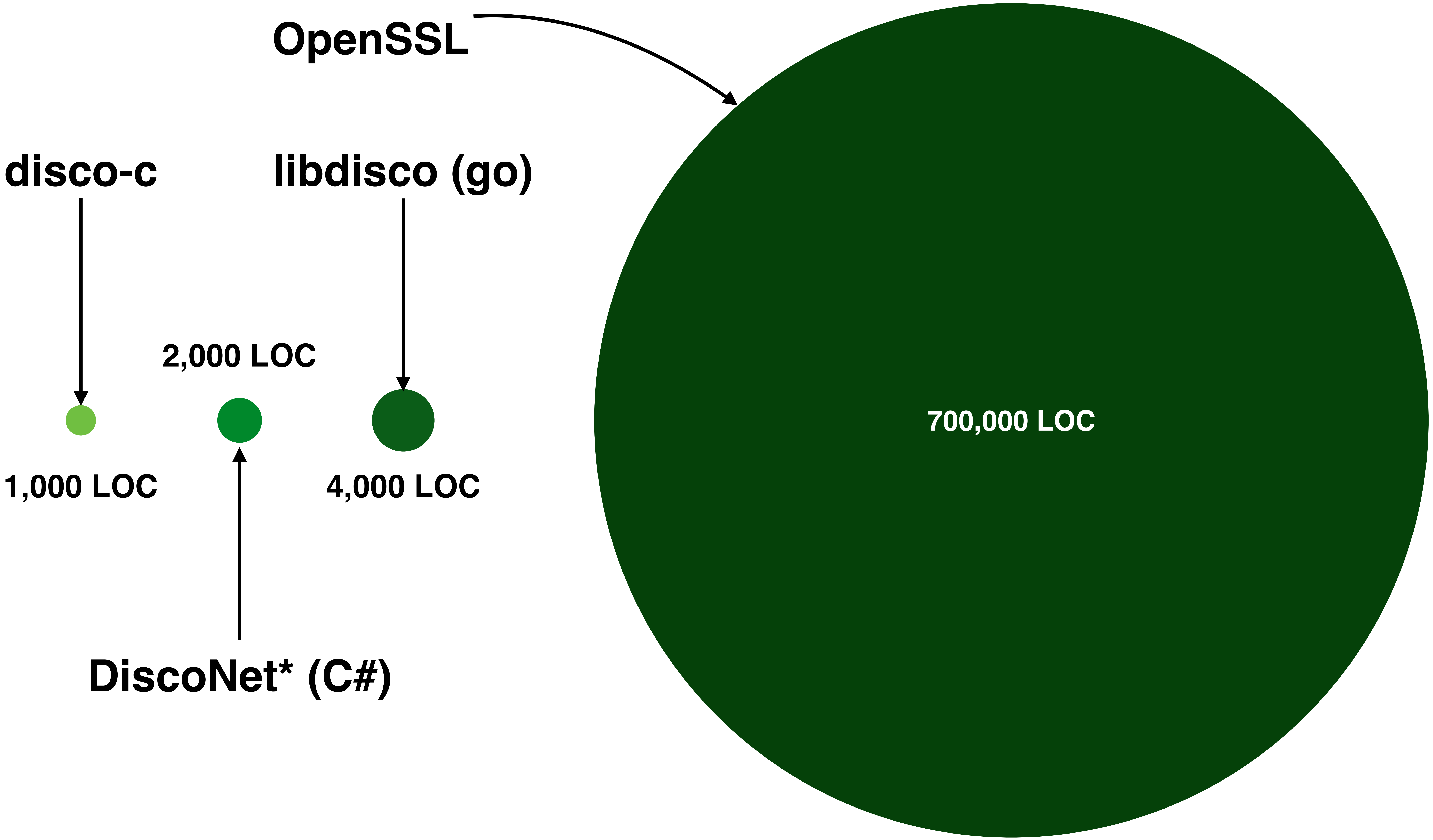
Designing cryptographic algorithms

Reducing a too large security margin

- Block ciphers: reducing number of rounds might be OK
- Obvious option considered by cryptanalysts
- Modifying other parameters: **doubtful**

Complex constructions with non ideal primitives

- Lose the benefit of an eventual security proof
- High risk of early broken versions (AEZ, Kravatte)
- **Require a large effort of cryptanalysis to obtain confidence**



David Wong, 2018

Motivation for BLINKER

Legacy protocols are unsuited for ultra-lightweight applications.

Academic research has focused on lightweight **primitives**, and suitable lightweight, **general purpose communications protocols** have not been proposed.

We need a generic **short-distance lightweight link layer** security provider that can function independently from upper layer application functions.

- Design with mathematical and legal **provability** in mind.
- Aim at simplicity and small footprint: use a **single** sponge permutation for key derivation, confidentiality, integrity, etc. (Instead of distinct algorithms.)
- Use a **single state variable** in both directions, instead of 8+ cryptovariables.
- Ideally this protocol would be realizable with semi-autonomous integrated hardware, without much CPU or MCU involvement.

Security Goals

Protocol designers should have provable bounds on these three goals:

priv The ciphertext result C of $\text{enc}(S, P, pad)$ must be indistinguishable from random when S is random and P may be chosen by the attacker.

auth The probability of an adversary of choosing a message C that does not result in a FAIL in $\text{dec}(S, C, pad)$ without knowledge of S is bound by a function of the authentication tag size t and number of trials.

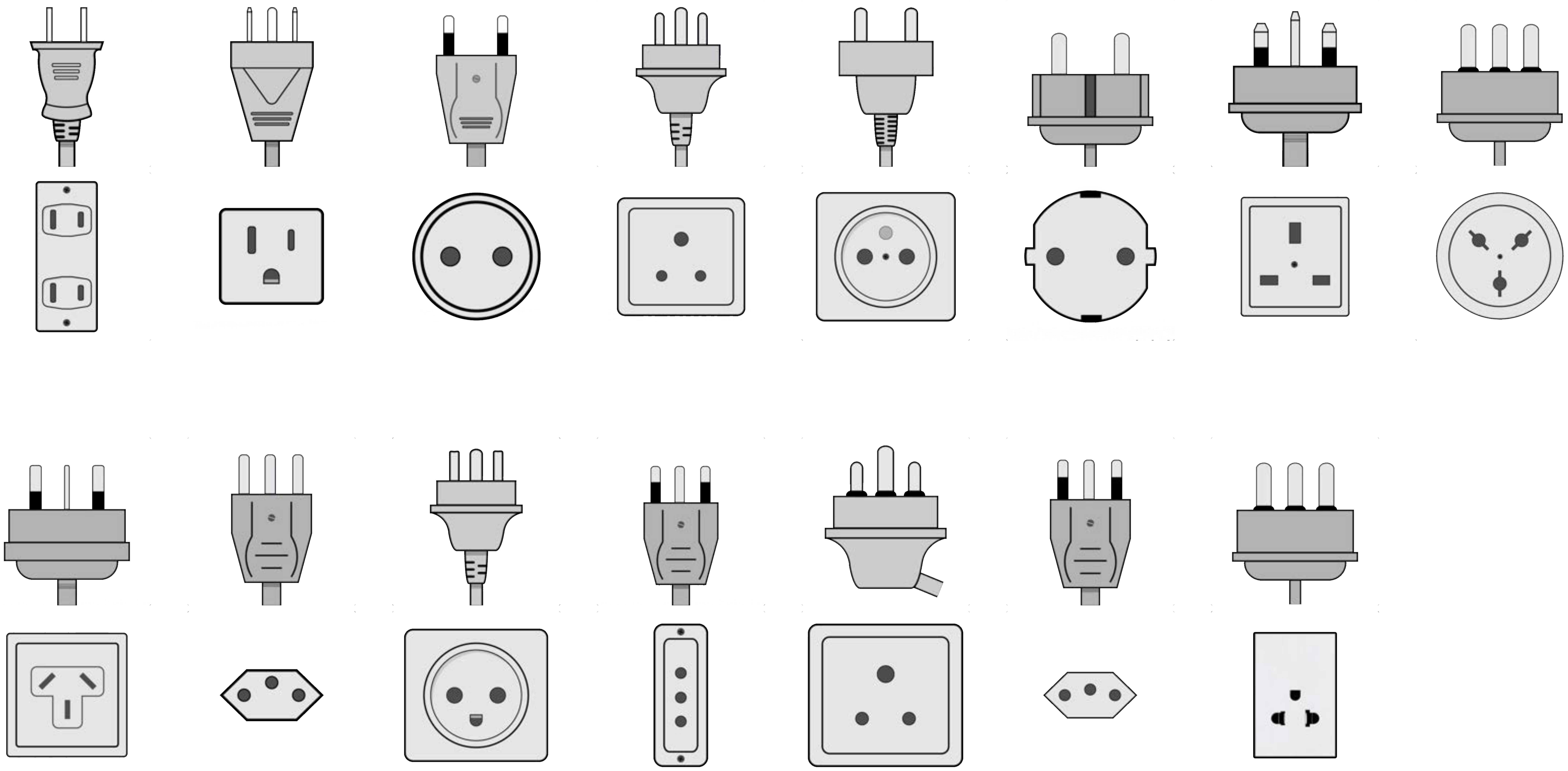
sync Each party can verify that all previous messages of the session have been correctly received and the absolute order in which messages were sent.

First two are standard Authentication Encryption requirements, the **last one is new**.


Protocol Builder's
Workbench

A couple of suggestions:

- Simplify the suite to one AEAD + one XOF
- Discourage shorter tags (forbid tags <64 bits?)
- Define 32-bit ‘tweak’ for key/nonce/XOF/…
- Ensure parameters afford extensibility (d, $h|t$)
- Let’s have a protocol effort (cf. AES modes?)



IEC, 2021

Linux Nordwalde, 2007

Cryptographic competitions

An illustrated history of Ascon

Real-world challenges

Lessons from usable security

Back to the future of PBC


hello@arnepadmos.com