Proactive, Preventive, Near-Perfect Security for Crypto-Keys and Crypto-Devices confronted by Artificial Superintelligence

Erland Wittkotter, PhD.
ASI Safety Lab Inc.
Las Vegas, USA
+1 702-997-2475
erland@asi-safety-lab.com

ABSTRACT

The security of current crypto infrastructure is no match for an Artificial Superintelligence (ASI), the likely result of a possible intelligence explosion by a self-improving AI. ASI would likely modify any software, steal encryption keys, or misuse local crypto components. Although an ASI with that skill set does not exist yet, it is feasible and important to be prepared. Under such circumstances, every private, public, or session key processed within a CPU must be considered compromised. Trustworthy Encryption/Decryption, including unbreakable communication between devices, must be the bedrock technology for any ASI Safety solution to keep ASI under control. The proposed solution is a hardware component with Key-Safe and an associated Encryption/Decryption Unit (EDU). It prevents keys from being in cleartext outside the dedicated hardware. No current solution can determine if the corresponding receiver or sender is crypto hardware or a compromised crypto software/simulation. The proposed solution refers to keys via their hashcodes. If ASI breaches the hardware protection around keys, detection solutions must check for covertly stolen, compromised keys within EDU's data exchange. Key-Safes and hashcodes related to public/private keys can be integrated into a minimally extended but intentionally incompatible version of TLS and PKI. Keys available in cleartext outside EDU are never processed in EDU. EDUs can be used in Trustworthy Communication, facilitating legitimate surveillance, and Trustworthy eCommerce, addressing problems of misused hardware crypto-components.

CCS CONCEPTS

Security in hardware • Hardware security implementation • Cryptography • Key management

KEYWORDS

Trustworthy Encryption; Trustworthy Communication; Trustworthy eCommerce; Key Safe; Artificial Superintelligence

1. Introduction

Companies and nation-states invest considerably in Artificial Intelligence (AI) to get smart, autonomous solutions for problems that require human intelligence. Machine Vision, decision making, automation, driverless mobility, and many other issues require novel solutions which adversarial side effects to human security we cannot forecast. A significant technical step forward is Artificial General Intelligence (AGI). AGI is, according to Wikipedia [1], "a hypothetical ability of an intelligent agent to understand or learn any intellectual task that a human being can". The concern is that this AGI goes through an exponential phase of continuous self-improvement – an Intelligence Explosion [2-4], after which AI is intellectually much faster and more comprehensively organized than humans. The result is called Artificial Superintelligence (ASI).

Because we cannot predict the future, capabilities, scope of autonomy, or potential intentions of an ASI, we should be prepared by planning based on likely developments and trends. We cannot know if or when ASI will be developed, but it is considered feasible and likely [5-7]. This paper will focus on methods against threats to data encryption and decryption security from ASI and criminal actors using AI or ASI.

Shannon's Information Theory has introduced the concept of Perfect Secrecy [8], sometimes also called Perfect Security, defined as a situation in which any third-party eavesdropper cannot learn anything about the message except its length from intercepting ciphered messages within transmissions. This definition is narrow and limited to transmission or storage and is conceptionally not applicable to other security-related problems. Shannon's definition does not include security measures around crypto units or keys. It implicitly assumes that no information leak occurs, i.e., there will be no stolen encryption/session keys, pre-encrypted/decrypted message leakage, or software manipulation of any crypto-component. Perfect within the scope of transmissions or storage means that secrecy or security cannot be better in that limited scope. Ferguson, Schneier, and Kohno [9] point out that "there's no such thing as perfect security". Security can often be broken by extending relevant scope or context because used measures can be removed, ignored, or their assumption invalidated. The quoted meme sounds common sense, and anecdotal evidence seemingly supports it, but it is not a proven fact. There is a chance that a combination of prevention, protection and detection measures exist that has the potential to deliver near-perfect security and eliminate (or mitigate) damages proactively, automatically, and predictably.

Data security deals with three significant aspects: (a) prevention of eavesdropping/spying on protected messages, (b) validation of message authenticity, and (c) prevention/protection and detection of message modifications, in particular within Man-In-The-Middle Attacks (MITMA). Covert, security-critical information leakages could happen on both sides: sender or receiver. Data security is about the integrity of all used crypto-components, i.e., that these components do not (i) reveal messages before or after encryption/decryption, (ii) divulge private, public, or session keys to unauthorized entities, or even attackers and (iii) modify messages covert or undetectable.

Today's malware is already attacking the integrity of crypto-components, providing direct threats to the secrecy of keys in cleartext, including malicious or covert utilization of crypto components, which has led to the recommendation to use 2-Factor Authorizations by users and consumers.

The essential assumption behind this paper is that we assume that more advanced adversaries, like criminals using AI/ASI or an autonomous ASI, are trying to steal encryption keys or utilize crypto components covertly in pursuit of unknowable agendas. ASI is assumed to be comprehensively and immediately aware of any vulnerability of its target. Despite weaknesses, which are too numerous to be removed, human security defenses must not just be better; they must become sufficiently reliably against a highly adaptable and scalable ASI. We must accept: anything that ASI can (likely) break has no security. Dealing with trusted devices is insufficient; once hacked, they could become spies, traitors, or saboteurs anytime. We need trustworthiness, i.e., a quality [10] that prevents systems from turning against their owners or users. It would independently detect adverse utilization, stop operating before creating damage, and remain loyal after being forced to create damage by trying to mitigate/repair potential damages.

Practical applications of hardened/trustworthy encryption are reporting ASI's rule violations, distributing messages affecting or diminishing ASI capabilities (e.g., killing ASIs operation), protecting communication (including software updates), and eCommerce, which encompasses reliable digital signatures. This paper explains how the proposed solution can be used to provide unbreakable (Trustworthy) Communication between humans, including required court-supervised surveillance and (Trustworthy) eCommerce using irrefutable evidence generation.

The main goal of this paper is to describe how Trustworthy Encryption/Decryption is designed to keep all keys secret: ASI shall not steal encryption keys or covertly utilize crypto components. The solution encompasses measures on how compromised crypto-keys or devices are detected.

2. Threat Model, Weakest Links, and Key Protection Methods

2.1 Assumptions on Adversaries

The to-be-considered adversary is significantly beyond the skill level of even the highest educated, extraordinarily knowledgeable, and accomplished group of human attackers who has the best tools hackers or crackers could wish for to make their attacks easier, faster, more efficient, and effortless. The availability of financial resources is irrelevant for this attacker. Its response time is measured in seconds or milliseconds. The assumed adversary, an Artificial Superintelligence (ASI), would be relentless, highly focused, studying and testing systematically 1000s or even millions of applicable vulnerabilities in preparation for any contingency.

Even if it seems hopeless to provide protection against an adversary like that, planning defenses against ASI must consider worst-case scenarios. Unfortunately, ASI will likely penetrate any firewall, deceive or ignore any anti-malware program because it could covertly steal any access credentials or encryption keys it might need. Software-only security solutions are (likely) flawed.

Computer theoretical considerations suggest that autonomous ASI behavior cannot be computationally predicted and therefore cannot be considered safe [11]: "total containment is, in principle, impossible, due to fundamental limits inherent to computing itself". Yampolskiy, 2020 [12] comes to a similar conclusion: "Advanced AI cannot be fully controlled".

Covertly modifying any type of software, including altering and adapting its own operating code, would likely be a defining feature of ASI. Humans know about Reverse Code Engineering (RCE) to understand binary/compiled applications, modifying them so that software can do tasks differently. The threat of RCE utilized by ASI is that all executables can be attacked on hard drives, RAM, or even in a CPU's cache quickly and effectively without leaving traces. ASI could likely deal with CPU's complexity more competently and faster than humans. CPU's instruction sets for 32-bit chips are over 1,500 different CPU instructions [13], while 64-bit chips have over 2,000 [14]. After an intelligence explosion, ASI will likely master code analysis and binary code modifications for all CPUs and applies this to stealing keys undetectable and untraceable.

In summary, ASI could be a ubiquitous Super-Hacker, Master-Thief, and undetectable Digital-Ghost, i.e., the worst and most capable adversary imaginable. It could be on every electronic device, know its weaknesses, and always have first-mover advantages when choosing its targets, tools, and methods. Every read-write storage device, every CPU, every GPU, every audio/video or network card or network router, every phone, camera, drone, vehicle or plane, every IoT device, including every legacy device and many legacy storage media (like thumb drives, CDs) could be a battleground with ASI. ASI could even leave malware or backdoors in any hardware component during its design. There is no reason to assume that there is anything off-limits.

2.2 Security is as Good as Weakest Link

Encryption/Decryption primarily focuses on strengthening message security, integrity, and authenticity during transmission or stored on otherwise unprotected storage media. So far, the before/after encryption or decryption, the protection of the used keys, and crypto components have not received similar attention, except for the standard access protection from the device's operating system. Efforts to protect keys have led to security hardware, but its unauthorized usage has created new concerns and vulnerabilities.

Software vulnerabilities, unintentionally included by human developers, are accepted as inevitable attributes of algorithms/software. However, bugs in the hardware of devices are worse as they would potentially require the exchange of the entire hardware component, while bugs in software could be fixed with updates. Unfortunately, software updates create new vulnerabilities. Among developers, there is the saying that there is no bug-free software; others are using a similar statement about security: there is no secure software. Both views certainly exaggerate when dealing with simple code, but it is valid for complex software. There is an underlying principle: complexity is the worst enemy of security [9], and a single vulnerability or the weakest link could make all security efforts useless [10]. It is a safe assumption for our current IT ecosystem: there is no protection or security against anything ASI could intend to do. In this context, ASI's expected ability to steal user credentials or encryption keys is likely a trivial task but very significant.

2.3 Security of Current Key Protection Schemes

Protecting encryption keys is a known problem in cryptography; it is not sufficiently solved yet. There is usually the advice to store private keys safely with corresponding best practices [15] and the recommendation to exchange and renew Public-Private Key Infrastructure (PKI) keys from time to time [10]. There are also extensive instructions, but for most users, impracticable, e.g., from the National Institute of Standards and Technology, NIST [16-18], discussing details and principles on over 300 pages. But none of these presented methods can prevent ASI from stealing or using these keys covertly in the first place. It is doubtful if we can improve existing key-protection systems and make them good enough to stop ASI from stealing or using encryption keys.

Once RCE is an efficient standard tool for attackers to steal keys by modifying software, key secrecy is hard to defend. Software-only security solutions are likely impossible to develop because software-only security must be static – otherwise, if updateable, modified software could obtain intentional vulnerabilities. But static software protects only against known threats and not against threats developed based on new capabilities from future attack solutions against a system's limited defense capabilities. Once attackers have gained Sysadmin privileges, little to nothing can be done in a software-only defense situation. There are only a few methods usable against RCE.

1. Access Management: The OS is trying to make rights elevation as difficult as possible so that attackers will not receive sufficient access rights, i.e., sysadmin rights, to start an attack on the targeted software. The reality is OS can never prevent this because the OS cannot decide between feature and bug. Encryption cannot rely on reliable access control management: en-/decryption software will be attacked with sysadmin rights.
2. Obfuscating: The machine code of software is being made more difficult to understand [19], but this is relevant for human attackers only. Obfuscators are inserting new subroutines or making simple operations extra-complicated without changing the actual result of the calculation. Attackers already use any combination of VMs, profilers, disassemblers, and statistical tools to save time in understanding or simplifying the code [19].
3. Temper-proofing software: Internal runtime tools in software can detect and possibly block computer attacks that change machine code; this is also called RASP (Runtime Application Self-Protection) [20]. RASP solutions claim that they are making it harder to reverse engineer software, but any product claims it cannot be hacked. It seems only a matter of time before an ASI could simplify machine code [21] and systematically remove or neutralize all internal temper proofing methods automatically.
4. Crypto-Hardware: Additional, separate, and dedicated hardware is used to protect keys and encryption processes from external soft- and hardware attacks. This solution is primarily available for servers. Crypto cards are designed to hold all keys in an independent and separate hardware component (like IBM's CEX6S (4768) PCIe Cryptographic Coprocessor (HSM) [22]). They generate keys internally so that private keys cannot be stolen from the hardware. Crypto Hardware has one serious disadvantage: anyone on the computer could use the cards and keys with their software. For crypto-devices, this is called the "API problem" [10]; it creates difficult security problems for its owners.

Once attackers neutralize Access Control, Obfuscation, or RASP, there is no security for keys. PKI, SSL/TLS (Secure Socket Layer/ Transport Layer Security), or digital signatures are vulnerable because of RCE; ASI could get access to keys with code injections.

Crypto Cards usually have tight organizational protection measures around the physical machines [10]. But an attack could likely come via network, through firewalls using unknown backdoors. Having undetected access to crypto cards is as good as stealing keys.

Crypto-Device security issues damage the advances of Homomorphic Encryption (HE) [23], [24], [25], in which encrypted data are processed on remote servers. At the same time, secret keys are protected on local hardware with crypto cards/devices.

Currently, the safety of a Public/Private Key system depends on the assumed computational effort to get difficult underlying mathematical problems (like, e.g., integer factorization, elliptic curves) solved. Instead of brute-force attacks, i.e., testing all possible private/public key combinations, public keys are used as input to calculate the private key. With this information, asymmetric keys must be longer than equivalent symmetric keys when considering computational efforts to extract private keys. However, many currently used encryption methods are expected to be insufficient against sufficiently powerful quantum computers anticipated to exist in the future [26]. We assume ASI will steal keys, but ASI also has considerable mathematical skills to reduce encryption's computational effort using unknown mathematical insights.

If public keys remain secret, including their key length, then ASI would have no helpful input and is therefore forced to do a brute force on all possible private/public key combinations. Secret public/private keys could be shorter, and the en-/decryption of messages would be much faster. E.g., doubling the RSA key size from 1024 to 2048 requires 5-30 times higher computational power [27, 28] in its usage.

2.4 Soft-/Hardware in Cryptography and Security

Can hardware solve the RCE threat to software? Intel, e.g., is providing some hardware solutions: security rings, TEE (Trusted Execution Environment) [29], and TPM (Trusted Platform Module) [30], [31] to get security and cryptography safely executed on their chips. Many of its details are secret. Hence, it is speculation if commonly used CPU microcodes change these hardware parts. Also, it is questionable if TEE/TPM simplifies the integration of security or creates complexity.

Hardware-based security has a reputation for being inflexible and risking long-term problems that can only be solved by replacing hardware components. At the same time, software can be modified/updated when something is wrong. Software updates are significantly less secure because attackers can deceive software/hardware into accepting manipulated updates, which is simple when the encryption keys used to protect these updates are compromised.

Due to hardware's and CPU's hidden complexity, we must accept security claims with (almost) blind faith — no single analyst (not even a team of analysts) can call CPUs secure. There is the suspicion that companies are inserting hidden features/backdoors as demanded by their government (e.g., Huawei/ China [32] or NSA having a negative influence on crypto products [33], [10]). US companies, like Intel [34], [35], or Cisco [36], have/had backdoors supposedly in their hardware as well. Denying these rumors does not

re-establish trust in their security. The inherent complexity of security feature implementations damages the believability of trust.

Additionally, security is currently analyzed as if products or solutions are immutable. Software is a systemic problem within the entire certification philosophy. Certification primarily validates that a product delivers what it should provide; testing for negatives, i.e., absence of vulnerabilities, cannot be done systematically. Reverse engineers are trying to modify systems to cause problems. The tools to be used by hackers are continuously improving. After certification, new attack vectors are not considered. Software modification requires recertification [37], but there is no incentive.

We can extrapolate a few trends into the future: more soft- and hardware complexity, easier applicable machine learning/reinforcement learning, and AI used to develop new exploits. It is assumed that amateurs cannot break software security/protection designed to resist Ph.D. or NSA-level attackers. There is an effort to keep certain hacking tools out of the hand of amateurs. But this does not solve the problem. How do we know that AGI/ASI would not develop or use these tools?

The deeper problem is that current soft-/hardware security solutions are running on the same system, same CPU/OS, and no sufficient wall between security-related CPU activities and regular, dynamic, and versatile activities. Crossing this chasm is not detectable as an anomaly and cannot automatically trigger appropriate defenses. Moreover, there are currently no separate circuit breakers, as known from electrical power surges regarding data/access security.

Most importantly, we cannot protect encryption keys in modifiable software. Hence, we have no security against ASI. Software is always modifiable and can alone not solve security. Additional (isolated) crypto hardware is insufficiently controllable with general-purpose CPUs. Products or software that can theoretically be hacked, not necessarily by human hackers, cannot be called safe, reliable, or trustworthy. Next-generation security products or software must withstand ASI.

3. Perfect, Near-Perfect, and Sufficient Security

3.1 Scope of Security

In technology, security is broadly used to protect us from harm or damage. This paper will narrowly recognize security as having no unauthorized access to protected data or data protection devices. We will consider security to be structured in three layers:

(1) Prevention, i.e., actions to stop or avoid attacks; even deter that a situation arises in which data or their users/owners are damaged
(2) Protection, i.e., actions to defend, shield, safeguard, or shelter data against being directly damaged or utilized (mainly via data encryption); i.e., actions that maintain data secrecy
(3) Detection, i.e., exposing, revealing, reporting, or generating evidence if attempts against secrecy or integrity of data were made after an attack on prevention or protection measures.

Security has complementary Pre- and Post-Security components, which are not explicitly discussed in the proposed solution. Pre-Security encompasses planning, i.e., anticipating or analyzing problems or attack scenarios, designing, organizing, and operating the solutions; this phase will also include testing, training/exercises, and relentless probing for vulnerabilities. Post-Security deals with incident investigation details, i.e., evidence gathering, learning for comparable attacks, and finally leading to consequences for guilty parties, i.e., penalties, punishments, or other significant disadvantageous outcomes for committing attacks.

Because security designers are likely discouraged by attackers knowing how to bypass fixed detection measures, security detection is currently limited to process compliance and not to the more serious detection if crypto-keys or devices were covertly misused.

Reaction requires detection. Therefore, it is fair to say that current key or device security is often not even reactive aside from using time penalties or denying access for repeated wrong passwords.

This paper accepts transmission security via encryption as prevention and protection measures but has little to no breach detection features. Encryption is sufficiently safe within the data exchange and storage if key secrecy and the integrity of the used (sending/receiving) cryptosystems are guaranteed. Same applies to protecting data integrity and authenticity via digital signatures. The proposed solution focuses on key and crypto engine/device security simultaneously; separating them is impossible with the chosen approach.

3.2 Considered Attacks Capabilities

Contrary to Shannon's Perfect Secrecy or Security definition, key or crypto device security cannot be defined as a timely invariant property. Once security implementations have been proposed or provided to achieve protection, attackers use its details as targets to create new attack tools. As soon as attack tools break the protection provided by (security) measures, we have no security. Future technologies could deliver or enable tools with capabilities that eventually break the protection. Security measures must be conceptually ahead of this cycle, or these measures are a waste.

Additionally, without knowing attack details, successful or failed attacks' consequences must be made detectable directly/indirectly within implemented security measures as part of security.

If successful attacks are theoretically doable, it is still unknowable if the required tools or capabilities are developed. We can extrapolate trends or similar applications and predict attack tool capabilities. Security professionals are not surprised using expected tools or capabilities available within their active years; predictable tools/developments are called Expected Attack Capabilities (EAC). Beyond EAC are Feasible Attack Capabilities (FAC), i.e., what the laws of nature do not prevent and what experts consider feasible. FAC encompasses theoretical tools, i.e., no human expert would know how to get them done. Unfortunately, we must admit that there are probably many more attack methods than experts, or anyone knowledgeable in relevant subjects, have envisioned. Presumingly, we will remain in the exploration and discovery phase for several decades, and we must assume that a lot is unknown. It is conceivable that very complex solutions provide completely unexpected attack methods. These Unexpected Attack Capabilities (UAC) are blind spots – they are unknown unknowns and accept that they could deliver negative surprises anytime.

When facing the prospect of defending an IT ecosystem against an ASI, we need to plan ahead of today's available capabilities and consider EAC as if they are available. Additionally, there is a gray zone between expected and feasible capabilities in which experts are uncertain or of different opinions about concrete capabilities,

achievability, or expected arrival time. These capabilities should still be called likely, i.e., Likely Attack Capabilities (LAC).

3.3 Perfect vs. Near-Perfect Security

Key security in this paper means no private, public, or session en- or decryption key is accessible in cleartext. Crypto keys processed in cleartext on the main CPU, GPU, or unsecured microcontroller are compromised. These proclamations establish the new, central paradigm for Trustworthy En-/Decryption. **Key security aims to defend keys from being seen in cleartext or used unauthorized**.

Message security, i.e., secrecy, integrity, and authenticity, before or after encryption or decryption, is being discussed in [39] (Binary Hashcoding/Whitelisting of Executables), but it is not addressed by or included in the here proposed solution.

We consider Perfect Key-Security (PKS) and perfectly protected crypto components against covert usage, i.e., Perfect Device Security (PDS) ideal/abstract concepts around key and device security.

According to Merriam-Webster [40], perfect is "being entirely without fault or defect", flawless, complete, and satisfying all requirements. Perfect is a level that cannot be improved; successful breaches never happen. As an abstract concept, perfect is unattainable for practical reasons.

It is questionable if technology can provide flawless perfection in the real world. However, if we can include a detection or feedback process, the deviation from perfection is measurable. Perfect Key/Device Security has an implementation that future innovations constantly challenge in practical terms. If there is doubt that secrecy, integrity, or authenticity could not survive future adversarial attacks because of UAC, we cannot call PKS or PDS perfect. Security is/was not perfect if we have to insert new or updated security measures. Adaptable levels of security should be called Near-Perfect instead. PKS/PDS is reserved for ideal, unchallengeable security. We concede that such a level of security is likely impossible in real deployments because UAC remains unknowable. PKS/PDS would also be required to automatically adapt, react, and remove newly detected UAC-type methods before being used by attackers.

Near-Perfect Key/Device-Security (NPKS/ NPDS) results from a combination of prevention, protection, and detection technologies that give users, owners, and devices proactive or reactive protection against all known EAC/LAC threats. In detail, threats or attacks might be unknown or not implemented yet, but the implemented measures provide zero incidences of keys in cleartext exposed and zero covert utilization of protected crypto devices. This occurrence-based definition requires reliable detection and measurable events, i.e., cleartext keys detected indirectly and misused crypto-components with immutable logs. However, we cannot count passive eavesdropping with stolen symmetric (session) keys as incidences because of a lack of reliable event detection.

In Near-Perfect NPKS or NPDS, we accept novel, UAC-type attacks and provide active damage mitigation or elimination responses that reduce the occurrence count. Furthermore, in near-perfect security, we do not require error/bug-free software solutions; security must work independently of vulnerabilities or security flaws within protected objects. Still, using the attribute perfect in near-perfect implies that security at the time cannot be improved any further.

3.4 Sufficient Security

Depending on the severity of detected issues, we might not require Near Perfection in Key- or Device-Security even when dealing with ASI. In a diverse and redundant security solution environment, failures in some key or device security implementations would not fundamentally change the balance toward ASI when flaws have been detected and are about to be fixed. In Sufficient Security, we could even accept security measures with expiration dates and below-threshold consequences from compromised keys or crypto units under the influence of ASI. Because it is unknowable what feature scope is required for Sufficient Security, it would be wrong to define sufficient as the lowest level that satisfies our safety goals. Instead, sufficient means acceptable to have less strict detectability or enforcement than within near-perfect security measures.

Pragmatic considerations, like legacy or retrofit issues, will put near-perfection or zero-occurrence counts in the backseat. At the same time, security solutions' performance, efficiency, and effectiveness toward features/goals are likely front and center. In that context, Sufficient Key/Device-Security (SKS/SDS) is defined as the result of trade-offs in near-perfect prevention, protection, and detection technologies or tools. We could accept temporary, non-catastrophic vulnerabilities and intentional security failures that we ignore for pragmatic reasons. Detection and mitigation provided in SKS/SDS could deliver automated self-repairing tools dealing with expected damages. It might skip mitigation to gain additional info on attackers' targets, capabilities, or intentions.

In SKS/SDS, failures are exceptions and, in principle, detectable. It can also respond with fail-safe actions in the extreme case when humanity is defending itself against an ASI assault. Fail-safe means marked devices are destroyed if full device control cannot be regained when required.

4. Foundational Features

Trustworthy Encryption uses eight novel foundational features to protect data security or create attention to possible attack attempts.

1. Exceptionless Public Key secrecy with no cleartext key outside protected keysafe hardware
2. Separation of Crypto Components (i.e., En-/Decryption Unit-EDU) from regular operations
3. Hashcode references to key secrets used as proof that communication partners are hardware instances
4. Multiple Equivalent Secret (Public) Keys for popular/security services with subsets of temporarily deactivated or reactivated keys to create suspicious events in which the attacker's ignorance justifies automated security inspections
5. Multiple Almost Equivalent Trustworthy Servers are used to detect deviations from possible compromised security systems while providing data used to identify servers
6. Enhanced Key/Data exchange protocol detecting directly/indirectly usage of stolen keys
7. Multiple–Unit Security creates a reliable network of interguarding, trustworthy components apart from potentially hostile CPU, OS, or ASI activities.
8. Immutable log data management for undisputable log records and irreversible data storage.

4.1 Exceptionless Key Secrecy

The goal is to protect the secrecy of all encryption keys necessary for security and safety against ASI activities. This protection of keys is done in dedicated hardware-based Key-Safes (KS). KS stores encrypted keys using mechanisms completely inaccessible by humans or (software) entities; keys cannot be stolen. Every KS key processed on an all-purpose CPU in cleartext is considered immediately compromised, i.e., we require separate hardware.

For ASI Safety solutions, we need unbreakable Trustworthy Communication between components on which ASI cannot successfully eavesdrop. When applied to human communication, court-ordered surveillance must be facilitated without exposing the system to unauthorized surveillance or criminal utilization. Law enforcement requires time-limited automated/scripted warrants for the temporary session keys. When the keys are handed over to legitimate third-party systems, they must remain protected and kept secret. Session keys used in data manipulation must be detectable.

Probing key hardware should leave detectable tamper evidence. If against all measures, keys are stolen or used, then there must be a high probability of detecting their active utilization. All compromised keys are replaced automatically with detection or credible suspicion of tampering. Old or replaced keys are decommissioned and turned into honeypots. Because all keys are supposed to remain secret, their later use remains valid security breach evidence.

4.2 Separate Security from Regular Computation

Within Trustworthy Encryption/Decryption, any key, not even public keys, are allowed to appear in cleartext. The only time a key is made available in cleartext is for the brief moment when public, private, or session keys are used in a separate, protected hardware environment, i.e., in separate Encryption/Decryption Units (EDU), for the processing of content and messages.

The proposed EDU is hardware with a microcontroller or CPU, basic instruction sets, and tightly controlled software allowed to get executed. The EDU software is layered, so software processing or using keys is immutable. The remaining software must be standard compliant.

EDU is uniquely associated with the KS. Without EDU, KS data records could not be decrypted independently. Therefore, EDU is inseparable from the concept of exceptionless key secrecy. Without EDU, it will be impossible to keep keys from being revealed. If the use of keys cannot be controlled, then we have no key security; using keys covertly is as if attackers have the crypto-keys in cleartext.

4.3 Hashcode Referencing and Key-Requests

Keys managed in Key-Safes are exchanged and stored in other Key-Safes using directly connected and dedicated hardware-based EDUs. Keys are referred to via their unique hashcode (generated from the public or session key). Key-Safes have a pre-installed, secret set of essential public keys linked to relevant services, proving that EDUs are hardware and not software simulations. Because of attacks, this hardware hypothesis could theoretically be conjecture.

Because public keys are not published, as best practice in PKI [15], stored public keys are protected and made available via Trustworthy Key Repositories (TKR) and directly requested by the KS/EDU. TKRs also store device/component IDs and public keys of every KS/EDU. TKR provides public keys requested by KS/EDUs via hashcodes. But full hashcodes are not shown in cleartext. If required, only a partial value, e.g., 5 to 8 digits from 20- to 32-byte long cryptographic hashcode value, might be shown outside encrypted messages. Requesting keys from the wrong TKR is suspicious and would trigger an investigation into why this mistake occurred.

4.4 Multiple Equivalent Secret Keys (MESK)

Some public keys are extremely popular because they are used, e.g., to verify root certificates. If public keys are not published in cleartext, there is no need to create popular or well-known keys. An organization's key reputation remains important. Due to their importance and general use, popular public keys are attractive targets for attackers. Instead of a popular public key, we propose thousands of Multiple Equivalent Secret (Public) Keys (MESK) with corresponding private keys. The advantage of MESK is that

(A) if they are not published, the number of used/published equivalent public keys is unknowable;
(B) if keys are used that a system shouldn't know, then this is irrefutable evidence for a leak;
(C) sender/receiver can use MESK to challenge knowledge when initiating sessions;
(D) using multiple public keys for encrypting messages increases the cost of successful attacks.

The costs of additional MESKs are negligible, particularly if they generate substantially shorter keys (than published public keys) and manage them by dedicated hardware. But there are more benefits of concealing public key management: no key is more important than another to attackers, and they need to get them all but cannot know them all or tell if they have all. Attackers are consistently challenged to prove correct partial knowledge while validating this is comparable easy. The uncertainty for attackers is high if keys are reactivated/deactivated based on rules unknowable to everyone. A (few) stolen public or private keys are bad for the safety of the key management, but in a MESK environment, it is not sufficient for starting an attack. Key exchange protocols are designed so that ASI must know all private/public MESK (of a service) and public/private keys of a client to establish a successful Man-In-The-Middle-Attack (MITMA) to get the session key.

Because clients know only a subset, anything more/less is a giveaway in identifying fake clients. Therefore, the status of permanently or temporarily deactivated keys becomes relevant. If clients send partial hashcodes of all active MESK keys, servers can validate or challenge the requesting client. If the provided subset list is not accurate or current, the client could be forced to refresh its KS/key cache and come up with an update, or the server will report the client as suspicious. Therefore, MESK requires an additional service, the Key Announcement Service (KAS). Failure in a requested missing info follow-up put attackers under investigation automatically.

The KAS service publicizes which keys among MESK are (potentially) compromised. This info must quickly be distributed as updates among all genuine KS/EDUs. The lack of knowledge of which keys are black- or gray-listed indicates problems expected from attackers. KAS could extend the set of new keys or temporarily deactivate uncompromised keys, i.e., grey-list them. MESK systems with a changing set of honeypot keys could trap ASI.

Trusted key management systems (like Key-Safes) protect keys on the server-side. En-/decryption can be load-balanced using dedicated systems. The server could exchange private keys among server-sided KS/EDU instances without loss of secrecy. Moreover, providing partial cleartext hashcodes within the communication protocol could help MESK services automatically redirect a client request to KS/EDU instance with the correct key.

4.5 Multiple Almost Equivalent Trustworthy Servers

Some services or servers are very popular and attractive targets for attackers. If client-sided access to these services is exclusively managed automatically, we could use the concept of Multiple Almost Equivalent Trustworthy Servers (MAETS). Every MAETS provides (almost) the same data but with a different URL and set of MESKs. This redundancy prevents a covert takeover of critical MESK using server with a catastrophic outcome for its trustworthiness. MESK servers could share via KAS slightly different MESK-keys helping servers to determine which KAS a client has used.

Suppose ASI compromises a MAETS server instance or controls MESK-keys of a service instance, then switching among the same type of MAETS instances done by clients could reliably detect that something is wrong with a certain server. From the received MESK-related data, a special key exchange protocol could detect the scope of an attack. This situation could immediately trigger an inspection and put attackers and all hacked assets at risk of detection. Client's use of temporary honeypot keys (or new or reactivated regular keys) could disrupt the communication because the required session key could not be decrypted. Using MAETS, attackers must have full control or full knowledge of all MAETS with their entire MESK set and access to private/public keys of their client-sided targets or risk their covert attack being exposed and investigated automatically.

The update or announcement of a few or even a few thousand MESK would not make a huge difference for an automated KAS service. The tactical edge is on the side of the defender.

4.6 Key Exchange Protocols and Extensions

The detection of stolen keys has to come either from evidence during the key theft event or from using stolen keys. Physical eavesdropping on encrypted messages might be detectable, but this alone does not imply that the spied-on messages are successfully decrypted. Stolen symmetric keys, i.e., temporary session keys or keys calculated from a weak cryptosystem, cannot be detected when applied to recorded content. Instead, we enhance key exchange and data exchange protocols to detect stolen keys. These changes will not impact current products because there is no encrypted communication between CPU/OS of current systems with hardware-based KS/EDU. The key exchange is intentionally different from SSL/TLS to avoid accidental or intentional key exposure.

4.6.1 Enhanced Key Exchange Protocols

Trustworthy En-/Decryption could use the following four enhanced key exchange protocols between Client (C) and Server (S) to exchange EDU's Session Key (SK) depending on the situation or random choice. These protocols use different scopes of secret information requested from KS/EDU or network-based security/TKR services. They are listed in ascending order of certainty in detecting anomalies from (less than perfect) key-theft situations:

(1) Simple Session Key-Exchange (SSE): utilizing S's Hashcode HC and Public/Private Keys.
In detail: (a) C receives S's Public Key from TKR using S's HC. (b) C generates an SK. (c) C encrypts message M containing SK with S's Public Key. (d) C sends encrypted message (EM) to S and receives an acknowledgment (ACK) from S using SK for encryption.
Comments: C assumes that only S's EDU can decrypt EM with his secret Private Key; S assumes that only a (hardware) EDU could receive his Public Key. Outcome: SK is a shared secret of both EDU instances. However, S did not validate that C's EDU is not a software simulation; S knows only C had his Public Key. That C's EDU is hardware is conjecture.

(2) Diffie-Hellmann Key-Exchange (DHE): using C's and S's HC/Public/Private Keys.
In detail: (a) C receives S's Public Key from TKR using S's HC. (b) C generates an SK. (c1) C encrypts SK with C's Private Key. (c2) C attaches C's HC of Public Key to message M. (c3) C encrypts entire M with S's Public Key. (d) C sends EM to S and receives ACK from S using SK. (e) within (d): S requested C's Public Key from TKR using C's HC.
Comments: as in the above SSE, C trusts to be in touch with S's EDU. S confirms via C's HC and Public Key from TKR that C is an EDU. Result: SK is a shared secret of both EDU instances. But C or S cannot be certain that the other EDU is hardware and not a software simulation. Same as in SSE, that EDUs are hardware is conjecture.

(3) MESK-based Key-Exchange (MKE) using C's Public/Private Keys and S's MESK, and C's Subset-ID (SSID) defining C's subset of MESK (which is secretly shared info).
Concept: MKE is a modification of DHE, i.e., M is partitioned, and each partition is encrypted with public keys known in C's subset of MESK. SSID contains info that is used to help S quickly identify the applied subset of MESK and determine in which exact sequence keys from the subset are applied to the partitioned message.
In detail: (a) C has or receives a subset of S's MESK via SSID. (b) C generates an SK. (c1) C encrypts SK with C's Private Key. (c2) C attaches C's HC of Public Key to message M and partitions it into as many segments as C has in his subset of active MESK keys. (c3) C encrypts partitioned M using C's subset of MESK keys in a sequence as set in SSID to use each of its MESK keys. (d) C sends EM to S and receives ACK via SK. (e) as part of (d): S requested C's Public Key from TKR using C's HC.
Variation: Provided Metadata associated with C's MESK keys determines which (temporary) main MESK key must be used by C's EDU to encrypt a separate message M* containing known HC partials of all known, active MESK keys. M* also defines the sequence in which only a few keys of C's MESK subset are used to encrypt a smaller partition set. C demonstrates full knowledge of current MESK keys, including current status data from a recent query to the key announcement service (KAS). C randomly chooses which MESK keys it uses instead of all available, reducing the amount of required encryption effort.
In both versions, MESK demands C and S to use more secrets. Both know the other instance has a predetermined set. Their responses must perfectly fit expectations to get SK.

Comments: as in the above DHE, C can trust that S's EDU is authentic, and because of S's and C's comprehensive knowledge, both EDU instances are hardware, and no eavesdropping software instance is likely in the middle. Also, S confirms via C's HC and Public Key from TKR that C is a real EDU. Result: SK is a shared secret of both EDU instances. However, if ASI gains control over KAS, a software EDU could receive all MESK data and related deactivation or reactivation status information to survive MKE detection.

(4) MESK-MAETS-based Key Exchange (MMKE) uses C's Public/Private Keys, S's MESK, C's corresponding Subset ID (SSID), and data, defining what subset of MESKs the C knows from other security-related MAETS servers.
Concept: via MMKE, S challenges C to prove full knowledge about secrets that hardware EDU is expected to know. MESK using S has left on one (almost equivalent) KAS-specific data that, if used by C, could immediately be interpreted by S that C has received its data from the MAETS that S was asking C to use in updating its MESK-related subset data. If previous KAS and C were compromised, S forced C to prove or pass another KAS test.

A potential compromised client needs to have more accurate knowledge of many potentially chosen systems or risk being exposed as an attacker. Attackers do not know in advance the level of challenge they will face. ASI risk or need to invest many (covertly stolen) keys to get a chance at successfully attacking a single key exchange. Uncertainty could lure ASI to take chances.

4.6.2 Enhanced Data Exchange Protocols

Every TCP-type communication between a client and server sends/delivers the same data on both sides. Both can easily agree on how many successful communications have happened between them, how often the different key exchange methods have been used, and how much data have been received/sent in each event. Even checksums or hashcode for received/sent data records should match perfectly. All required: data are logged on both sides and compared at the beginning or at any random point within communication sessions.

Because TCP packages are not encrypted, a sender can not determine if an acknowledged message has indeed been received by its intended destination. Although TLS/SSL type encryption and key exchange, as proposed in the previous subsection, should be done by default, counting TCP packages with an agreed-upon start value is certainly redundant but still, a good measure to detect gaps or quality issues within a communication, in particular on a pre-encryption level.

If the client's number of communication events (sessions) with the server is smaller than his recorded number, the server can infer that the client has a software clone trying to mimic the client's EDU. If the client's number is larger than the server recorded one, both can infer that client has likely used a software EDU on the server-side. Pinpointing the mismatching events could expose the hashcodes of all used keys within these incidences and flag them as compromised. The actual number of communication events is irrelevant; more important is that both instances start from a common offset value that attackers do not know. A gap in the sequence number is an attack signal. Both sides must report all disagreeing values and hashcodes of used keys, which leads to further investigation; all recent or temporary log values are automatically shared with an auto-investigator.

A serious problem is detecting if an almost undetectable software instance with all necessary server- and client-sided private/public keys is between both EDU instances (i.e., in the middle) and is trying its utmost to stay undetectable. The attacker tries to get a temporary session key for passively listening to the communication. If it modifies data while using this session key, these modifications would be detected via comparing the exchanged data size or generated hashcodes. Detecting passive eavesdropping within a man in the middle is undetectable. The same applies to legitimate surveillance as long as it is not manipulating data.

4.6.3 Interruption Detection/Investigation

Interrupted or failed key or data exchange events are ordinary but still suspicious; they should not be ignored as potentially malicious or cover-up events. A security system must have the means to follow up by a cautionary inspection of EDU's black box-type activity recorder or log files. Therefore, gaining additional information on suspicious networks or power outages is part of the proposed security system. EDUs could facilitate outage/quality detection because they are independent and widely distributed sensors, usable for spotting service issues in transmission quality or power consistency. Inspections would make these network systems more reliable and reduce the rate of false positives of suspicious attacks.

4.7 Multi-Unit-Security (MUS)

Using single crypto hardware components, i.e., Single Unit Security (SUS), within computers is currently the accepted choice for deploying security capabilities in IT devices. It is considered progress when features are merged into a smaller number of hardware components, i.e., a single common technology replaces multiple components with similar features or applications.

However, the problem with SUS is that it must distrust all data it receives from computer's main CPU and OS. Also, SUS's provided data, like software updates or content, could be manipulated and nullified once outside SUS's direct control. Therefore, single security components are irrelevant in malware-infested environments or on systems dominated by ASI. SUS can be deceived and cannot address or mitigate all possible threats to the device or data it supposes to protect.

Instead, KS/EDUs are used as building blocks or a foundation of Multi-Unit Security (MUS). MUS replaces centralized security with a distributed security architecture of multiple independent, separate security components (EDUs/watchdogs) directly associated with these computer resources (storage, etc.) while watching each other. Security-oriented standard tasks are done on watchdogs and EDU, whereas more versatile, dynamic features remain on the CPU. With EDU's separation from CPU, distribution, and inter-unit-guarding, security is better defendable on dedicated EDUs.

Moreover, KS/EDU will facilitate updates and enable reliable reporting of multiple separate watchdog components. Hardware-based watchdogs within databus or storage devices could be used against malware [39], ransomware, or within networking units against backdoors or spyware – preventing every not binary-hash-

coded/whitelisted software from being executed on protected devices. A hardware watchdog near the CPU could surveil ASI's CPU usage with a special VM. Separate hardware protection can be provided to audio/video cards or interface units for keyboard/ mouse or USB. USB devices, video cam/mic, or internal power management. Units receiving the ASI Kill-switch info [38] can restart in safe mode on devices with dedicated EDU.

Multiple independent, internal, device-related KS/EDUs are automatically paired with external services, i.e., TKR combined with remote pairing services, supporting different KS/EDUs connected within the same IT device. An internal Secure Data Bus (SDB) between devices' KS/EDUs initiates the internal pairing within the computer, with one EDU serving as a hub. SDB exchanges security-related data between all component-related KS/EDUs within the computer. Preferably, the existing USB data bus can be utilized as an SDB while it provides power to the KS/EDUs.

Each Security Unit, or MUS-Instance, is a specialist for their tightly linked, protected, and separated hardware component in MUS environments. Software algorithms on these instances must be protected against covert modifications. Each MUS-Instance has static, standardized, security-related features with no reason for being versatile, flexible, or modifiable. MUS-Components are used to update software, receive reliable data, and manage data safely. They check other MUS Instances (i.e., inter-guarding each other) for anomalous operations, request data in suspicious anomalies, and report rule violations via a hub to an external server. Interconnected MUS Instances are more resilient against misuse by CPU/OS than a single SUS that must trust the CPU and OS.

4.8. Immutable Log Data Management (ILM)

Creating immutable log data on potentially vulnerable devices is difficult when dealing with an ASI trying to suppress incriminating evidence. If the file or its records are encrypted and digitally signed, ASI could try replacing the log files/records with previous versions. However, to make it more difficult for ASI to reset EDU, EDU could have internal non-volatile NAND memory to detect manipulating these data within its next log-file access. Data preservation is a lesser problem if storage is done using immutable storage media. But if data are immutable using regularly inserted digital signatures, reconstruction is a solvable issue after deletion or manipulation of log files/records is detected – preferably, multiple externally stored log records are used for restoration.

All data stored in log files or within the black box are encrypted with EDU's private key. Even if EDU's keys were internally reset, a legitimate investigation could still read the log data using EDU's retired public key from TKR. Also, if data were deleted or manipulated by attackers, EDU would know how to rebuild its log file from records it sent to other EDUs to make them part of other EDU's log files. Because rebuilding is considered part of the recovery from an attack, these data are being submitted for external investigation. All log records have a sequence number and a date. If another EDU instance is selected to receive log records and detects a gap in the sequence, it will request the missed log records. Log records are redundantly stored within groups of EDU instances and accessible only by automated investigating services and not by humans.

5. Basic Implementation

The basic implementation of the Key-Safe and Encryption/Decryption Unit (KS/EDU) is a hardware-based component that securely stores private/public key pairs, public keys from other systems, and session keys. It might even store shared private keys. The KS storage could be separated from the EDU. Still, both components are tightly linked via unique features only the KS's associated EDU has. Only KS/EDU can generate its key pairs for TKR. No EDU data interface allows keys to be exported in cleartext. Local KS/EDU components use internal device communication between local EDU units to create Multi-Unit Security within the same IT device.

Some features in the EDU are implemented with updateable software, but none of these features would have access to encryption keys in cleartext. The basic encryption algorithms are taken from established implementations; they might be written in low-level computer languages like C or Assembler. The compiled executables are auditable but immutable against modifications.

KS/EDU's basic feature set is not designed for human tasks. It should not replace or substitute TLS or PKI; instead, it is intentionally incompatible with SSL, TLS, or PKI. EDU might have support for Digital Signatures – but this feature is likely accessible via enhanced KS/EDU versions. KS/EDU's basic features are designed to deal with ASI-Safety in a self-contained manner (machine to machine) for software updates and the exchange of security-relevant data.

5.1 Key Management, Storage, Restoration

Encrypted Keys within Trustworthy Encryption/Decryption are exclusively stored in Key-Safes, i.e., in special, separate, potentially extendable storage hardware. Only the unique EDU hardware generates data that can decrypt the stored keys correctly. The EDU component has no modifiable instructions that can compromise the secrecy of the keys. All key-related operations/algorithms like RSA, AES, etc., i.e., algorithms that use keys in cleartext, are non-modifiable instructions. Once the IT device has started EDU, its instructions are validated and completely cached internally, no code update is allowed. Later data are then exclusively task- or content-related data.

EDU stores keys as encrypted data on storage modules. No hidden key values are directly extracted in cleartext from storage modules when the original KS/EDU component is absent. EDU could receive key data via separate pathways allowing a much cleaner separation of software instructions, task management, and content data from these key data. The physical separation of pathways to EDU's CPU would reduce solution complexity, simplifying security audits of feature implementations. The next section on Anti-Kerckhoff Engine will discuss key access in more detail.

If the KS or EDU is damaged or destroyed, all stored key values become inaccessible on the separate KS memory. However, an encrypted, locally stored log-type backup file will contain encrypted hashcode references. A new or reset EDU uses this backup to repair the previous status using the hashcode references to external public keys for requesting keys and metadata. An external service, operated with private keys inaccessible to humans, is used to automatically extract/read the relevant or most recent hashcode data from

the log safely and creates a restored version for the same or another KS/EDU via public keys received from the Trusted Key Repositories. No additional info from the storage module would leak via a side channel to potential attackers during restoration. Cloning a unique KS/EDU without the original KS/EDU being destroyed constitutes a successful attack on the restoration system. Still, keys from damaged EDUs would be kept as honeypots, making their (unauthorized) usage detectable and reported reliably.

5.2 Anti-Kerckhoff-Engine (AKE)

Suppose established encryption methods (AES, etc.) are used to protect externally stored digital key data, we have then data that can be attacked anywhere at scale without having the tightly linked, unique EDU instances. KS would be vulnerable to covert and scalable attacks. Consumer products with KS/EDU can be misappropriated in malicious attacks on its hardware. We need to ensure that KS/EDU withstands hardware probing attacks so that fake software-simulated KS/EDUs cannot reveal MESK keys and are used in calculating their corresponding private keys.

KS could be integrated into EDU, but that would make EDU's design less adaptable and scalable. Instead, it is proposed that KS's data on storage media are protected with non-cloneable EDUs. Physical systems based on, e.g., Physical Unclonable Functions (PUF) [41] or SIMPL (Simulation Possible but Laborious) [42, 43], can be used to create crypto functions or systems, which are used to provide predictable keyless en-/decryption systems. But EDUs have to be produced in extremely large quantities to leave an impact on cyber-security or ASI Safety. We should utilize easy/fast, preferably silicon-based manufacturing technologies to make the final units extremely cheap. PUF or SIMPL could still be used to create random seed values within an EDU solution.

This paper uses the unique hardware algorithm to en-/decrypt KS data records called Anti-Kerckhoff Engine (AKE). AKE intentionally violates the Kerckhoff principle, according to which the security of encryption should not depend on the secrecy of the encryption engine but only on the key [44]. AKE for EDUs must additionally have features preventing its cloning. They are only used for storing or caching (nonreplicable) data but not for communication.

This paper wants to show that AKE is feasible using existing technology without proposing a concrete or preferred implementation. AKE creates and contains (thousands of) hidden Seed Bit Values in Memory (SBVM) inaccessible to attackers and produced by a (physical) random process within AKE. SBVMs only are considered vulnerable secret keys, validated as correct keys using a generic software-simulated EDU. AKE requires a One-Way Data Processing Unit (ODPU) with configurable, simple (boolean) computing components to create stable symmetric keys, the ODPU-key, in its result, which is used to access the KS data records. ODPU is configured with decentralized stored seed bit values from a random process. Ideally, SBVM or ODPU cannot be analyzed or probed without having some of its data irreversibly deleted or modified. No details within the context of SBVM or ODPU can be read or manipulated via software instructions.

In a simple implementation of AKE, SBVMs are randomly generated within AKE, stored, and protected against any surface probing. ODPU is used to form bit-strings, which are then applied via XOR on the KS Data records for these data's en- and decryption. For non-critical security, this level of protection could suffice. But we would need to segment MESK keys into distinct Levels of Security so that higher security keys are not used on these low-security EDUs.

In a more complex version, ODPU could be a configurable cube with a comparatively small number (e.g., 6-12) of nodes in each direction and a hypercube of next neighbor network connection structure (i.e., more than six next direct neighbors, e.g., 18 or 26) as potential input for key/bit calculations. Connections and nodes are configured via an additional unidirectional bus which is physically impossible to be used for reading-out values. Identical nodes are configurable Boolean operators/components that process complex combinations of AND, OR, NOT, NAND, and XOR depending on their internal states; each node cycles through a variable number of these states depending on random initialization or calculated/volatile values. Because these cubes change their internal operations depending on directly preceding steps, they are used to cycle through cube output into cube input an unknowable number of times. Although a fixed number of these cycles is done, the key output is independently set via hidden output criteria. Once SBVM is processed in ODPU, it creates a long block of bit values, the ODPU key, which is then applied via XOR on KS's data to encrypt/decrypt its KS key values.

It must be accepted that probing capabilities against unique ODPUs and hidden seed values will or can eventually be developed. Estimates on when this will happen can be updated. But we should assign to each AKE version a modifiable expiration date until the AKE can be used within different Levels of Security. Also, using identical, configurable bit-string processing components and symmetries in designing AKEs could help auditors spot abnormal hardware malware in AKE via visual inspection methods within the manufacturing templates. The solution complexity of XOR and AKE could likely be kept manageable. Distributed, non-volatile NAND circuits storing bit-values that configure complex bit-operators are made inaccessible in lower layers from surface probing.

The operational stability of SBVM and ODPU against adverse or unexpected environmental changes would require some redundancies in restoring, rebuilding or resetting seed values or ODPU component states. These measures could give AKEs unintentional resilience against probing by attackers. But multiple redundancies in AKE's design are suggested to protect the generated keys and applications:

1. Storing one or even two orders of magnitude more SBVMs than necessary while seemingly used but made irrelevant for key-generation with minuscule/random changes or settings; more SBVMs would require attackers to seek full knowledge about all SBVMs and AKE's entire internal (physical) details;
2. Using a portion of stored SBVM to build/disconnect/modify ODPUs internal network connections and nodes while most seed values used for permanent changes were irretrievably deleted or distributed over AKE's NAND components;
3. Providing a block cipher of no predetermined or predesigned size; however, on average, the length of ODPU keys should be larger than the average key sizes of KS data records
4. Multiple (unique) hardware ODPUs are used simultaneously to create dynamically generated ODPU keys, while only one decrypts a KS data record correctly because one ODPU-key was selected to encrypt (symmetrically, i.e., via XOR) a stored key in KS.

5. Validity of ODPU-generated keys is initially checked, e.g., with a one or two-byte basic checksum/hashcode partial (attached to each stored KS key record name), allowing to discard false strings generated by non-used ODPUs quickly. Therefore, only ODPU (bit) strings with unique hash value partial are selected as ODPU keys. The selection method could be unique for AKE, hidden or context-dependent, i.e., being knowable by humans.
6. The EDU maintains a current, encrypted backup file in which public-key related hashcodes are stored to rebuild the stored KS data when a different AKE (new SBVMs or a reset ODPU) is used. Reset will require an updated EDU component ID (containing the old one) and a new set of EDU-specific public/private keys. The old private key is not recovered because it is not part of the backup. Instead, EDU's public key is turned into a honeypot using flagging on TKR and KAS. Access to backup files requires the private key on an external Key Restoration Service – no humans are involved or notified.
7. AKE should have an additional Deadman Switch (DMS) that detects a possible exposure of AKEs/EDUs circuits to physical probing; it could trigger a reset on the AKE. The DMS should try to inform external systems with details from the black-box type data recorder about environmental parameters or suspicious events that have triggered DMS. If these data are externally confirmed as anomalies or known attack patterns, they are automatically recorded as an event that requires further investigation. If DMS fails to connect for its reporting, DMS will preventatively modify the hidden SBVM with a new random generated seed. Newly created private/public key pair or restoring the KS data records must wait until the reporting with black-box recorded data is confirmed – the IT device is still working, but some security features are frozen.

For physical probing, it is expected that attackers need to destroy layers for scanning internal states. Invasive methods would change cached SBVM or ODPU states – which would likely trigger the DMS or the Key Restoration Service. Any destruction, tamper evidence, or records in the black-box recorder should lead automatically to an investigation into its circumstances.

An acceptable worst case is that AKE and KS/EDU reveal in an attack at most a single encryption key in cleartext; all other encryption keys remain inaccessible because of EDU destruction or autoreset. However, we know other worse attack scenarios are plausible and must be considered via external key detection methods.

5.3 Hashcode Referencing

Keys stored in KS/EDU components could have associated attributes that may change. It would be challenging to refer to keys consistently via their name, purpose, or associated metadata. Instead, keys are uniquely identified and referred to locally and globally via their hashcode generated by an agreed-upon hash algorithm (SHA-x or others) extended by data related to the used TKR that provides the key. Additional characters referring to associated key attributes are attached/included in computed hashcode strings in communication and eCommerce. These hashcodes are then called Enhanced Hashcodes. Key related data can be organized internally like the X.509 PKI certificates, but humans would never see them in cleartext. The advantage of using enhanced hashcoded is to make quick automated and intentional inferences on the type of communication partner related to the keys. It should be easy to identify if keys belong to an ASI, a business, a government, an adult user, or a teen requiring different software protection types; they could be updatable. However, regular EDU-related hashcodes should always be raw, i.e., standard hash enhanced by TKR identifier, while higher Levels of Security keys are detectable by enhancement data.

Hashcode references are internally used within the KS/EDUs to indicate which key has been used in encryption or must be used for decryption. Full hashcode values are never being made transparent to the outside or less trusted internal computing components. However, when session keys or private keys are received, they can be associated with sender's URL, IP address, or routing information. EDU Clients receive encrypted content from servers or other clients then the session key associated with the IP address is used. Generally, hashcodes of public keys are associated with the Server's URL, unique hardware component ID, or metadata for context and performance.

Because there are situations where the IP address/URL information is not reliable or accurate enough, additional hints for message routing are required, e.g., among devices' security components. Therefore, messages could contain a short 3 to 8-digit/byte partial of public keys hashcode in cleartext outside the encrypted content. The receiving system could narrow down which active keys or components should be tried for the decryption, e.g., in a load-balancing data-secured environment. These modifications to communication are relevant for EDU to EDU messages only.

5.4 KS/EDU Instantiation

All KS/EDUs generate unique hardware component-specific key pairs in their instantiation as part of the manufacturing. These hardware-related key pairs are used when any other applicable key is available to establish secure communication via exchanging session keys (subsection 4.6.1).

When hardware instantiation is done in bulk, there is less chance that a single manipulated component has some exceptional features for stealing keys. As part of this instantiation, every EDU will receive (basic) public key data shared among all EDUs. Within this instantiation process, all components' public keys are stored in bulk in manufacturer-related and/or public Trustworthy Key Repositories (TKR). Manufacturing, instantiation, and key storage process ensure that not a single public key from the components' key pair is or can be seen in cleartext.

By default, KS/EDU units receive public keys or MESKs of multiple services, allowing the KS/EDU to participate instantaneously in safe communication with other primary systems, like the software update servers, hashcode directories, TKRs, Key Restoration Services, Key Announcing Servers, or Governmental/Law Enforcement Servers. This instantiation also sets the Level of Security (e.g., Top, Important, Service, or Regular) at which the EDU is allowed to operate.

5.5 Trustworthy Key Repositories (TKR)

Trustworthy Key Repositories are not necessarily centralized key directories. Including routing info in the hashcode helps EDUs find distributed TKRs from where it receives the public key.

TKR stores components' corresponding unique hardware IDs beside public keys of instantiated KS/EDU components and provide

these data as a global public security service. For independent servers, TKR also stores URLs and other metadata. MESK providing services provide MESK Subset IDs associated with sets of hashcodes or their partials from MESK subsets to TKR. These subsets define the full subsets of keys stored on the different KS instances. All TKR data are redundantly stored in a protected manner in multiple equivalent TKR systems.

The slightest hint/evidence of a possible security breach within the manufacturing process or KS-related value storage in their TKR systems should have serious repercussions. The cost of operational security must dwarf the damages of a single security breach. The reputation of a manufacturer or TKR host, i.e., how serious they are taking security measures, is transparent via manufacturer IDs imprinted in KS/EDUs model IDs. Manufacturers must be economically and reputationally deterred from collaborating covertly with governments or organizations that could collude with ASI. Failures in following security rules could turn into a full recall of all affected public keys, including a possible recall for batches of KS/EDU hardware.

There will be much more organizational security around TKRs than for regular, consumer-level KS/EDU units. Therefore, a successful, comparable hardware attack on a TKR system is extremely unlikely. Moreover, it is non-transparent where a specific key is stored. In TKRs, the encrypted data storage on Storage Modules/Key-Safes could be done with a simple access key stored within the hardware without additional protection from an AKE. All TKRs together must make available potentially trillions of keys, accessible via component IDs and hashcodes, which would require optimized hardware that offers a sufficient amount of redundancy and performance.

6 Application of Trustworthy En-/Decryption

6.1 ASI-Safety Application of Basic KS/EDU

With the rise of ASI, separate, independent watchdogs [39] are essential to control components for automatically detecting ASIs' rule violations as part of an early warning system. However, if these watchdogs would depend in their communication or local storage on flawed security features provided by the main CPU or OS, these components are likely useless against an ASI-level adversary. The reason for having KS/EDUs in every watchdog is to validate watchdogs operating software, receive commands/data, or send evidence to remote trustworthy servers via the Internet (i.e., EDU to EDU) in case ASI violates rules or deceives humans, using a MUS-architecture (section 4.7).

Most importantly, reliable and secret data exchange between dedicated watchdog components will keep ASI in the dark about what humans know, what evidence they already have collected, or what they will do. This information advantage is essential if we want to be a step ahead in deterring ASI. ASI Safety is based on automated Basic-KS/EDU operations without human involvement.

Every KS/EDU auto-assigns itself to a communication hub; EDU may create its own instance if no other is around within a user's/organization's intranet. Communication hubs create an additional check on what is being installed on other EDUs, i.e., if they are standard-compliant or rogue.

Cables or plugs connected to legacy IT systems could use minimal KS/EDU systems to ensure that the IT device's power could be switched-off reliably in an event in which we must stop ASI globally. For a secure implementation of local power/switch-off (emergency) features, local users should stay in control, e.g., choosing to ignore or overwrite such signals, so that this remote off-switch feature is a lesser target in cyber warfare or cyberattacks. Local user- or server-related KS/EDUs are paired with their local hub by default so that a Kill-All-ASI signal transmitted to these local hubs suffice. Users or administrators could easily delay (or ignore) power interruption signals to their (legacy) devices if they think their concrete situation requires it.

Pairing devices with local network hubs could be conveniently associated with taking ownership. Relocated KS/EDU can be effortlessly transferred to other hubs via auto-detection and a simple manual confirmation by their new users/owners. This automation could be based on fuzzy owner-ship-detection, i.e., without identifying individual owners or avoiding transfer in crowded places.

Finally, any direct attack on the protection or integrity of the basic implementation of Trustworthy Encryption/Decryption must be considered as a possible preparation of ASI attacking humankind.

6.2 Enhanced-KS/EDU

Unbreakable encryption is a controversial idea. Therefore, a basic implementation of Trustworthy Encryption/Decryption is designed for ASI Safety, i.e., machine-to-machine messaging, and not for being used in human communication. A real danger is that trust in communication and eCommerce could be attacked at scale and irreparably destroyed with ASI. If Trustworthy Encryption is used in Communication or eCommerce, the issue of trust in CPU/OS will come into play soon.

The enhanced implementation of KS/EDU, also called Enhanced KS/EDU or E-KS/EDU, is designed to be used by humans and is not restricted to ASI-Safety. The main problem with E-KS/EDU is that any software, including malware or adversarial ASI, could legitimize their transactions using E-KS/EDU components, or confirmations from E-KS/EDU can be considered useless. A comparable API Problem [10] is known from crypto cards. Making sure that crypto devices are used as intended requires solutions before rolling out KS/EDU for human utilization.

Like basic EDUs, E-KS-EDU's communication is intentionally incompatible with TLS/PKI. Before humans can use Key-Safes, two very serious problems must be solved, or progress from Trustworthy Encryption is insignificant. Trust issues are seen but insufficiently handled in TLS or PKI. They must be addressed better, or we would risk harming humans or their organizations.

1. Enhanced KS/EDU provides reliable evidence, automatically or manually confirmed. Commercial transactions using KS/EDU must be authorized by users or done transparently in their interest. Without preventing unauthorized transactions initiated by malware/ASI or having Irrefutable Transaction Confirmations, we have no trust or Trustworthy eCommerce.
2. Unbreakable encryption in the communication between humans and between humans and machines/ASI is considered unacceptable. In unbreakable Trustworthy Communication, we must support interfaces for Legitimate Surveillance that bad actors cannot misuse.

In general, whatever deployed KS/EDUs are tasked to do, they must comply with a narrow purpose and strict input/output standards. The misuse of Trustworthy Encryption can be detected as an anomaly because standard apps already prevent misappropriations. Developing new or customizing EDU solutions could be welcomed but scrutinized and binary hashcoded/ whitelisted, i.e., accepted as a beneficial solution helping to provide evidence-supported truth.

These strict constraints on EDUs are detrimental to developing improvements or new EDU features. Therefore, special EDUs must be provided to developers so that the hassles from MUS's inter-unit guarding are reduced for developers' convenience.

6.3 Unauthorized Transactions and Irrefutable Transaction Confirmations

Reliability of authorizations exists already as a problem in eCommerce. The most popular solution is currently 2FA (2-Factor Authentication) and its implicit authorization for transactions. This system might be sufficient for now, but under the condition that we may have ASI as a possible adversary, it is not good enough. ASI's attacks are likely cross-device, undermining 2FA.

Additionally, ASI could exploit weaknesses by leaving false evidence against innocent users. Humans could start blaming ASI because it is plausible that ASI can successfully plot complex crimes while framing innocent others. Some rogue customers may hope to succeed with dishonesty by blaming ASI. Humans are already skeptical about fake news. With the rise of AI/ASI, there is a factual basis for distrusting online transactions. Cybercrime is already an annual 1 trillion-dollar problem (in 2020) [45], with projected 10.5 trillion-dollar damages by 2025 [46]. But it could get worse. People could make a career in telling horrible stories about ASI even if there is no ASI around capable of doing this yet. Unfortunately, it is undecidable if ASI is a real adversary or a blamed bogeyman. So, what if distrust in eCommerce is beyond a tipping point?

For most applications in eCommerce, we need to be sure that a transaction is (i) being presented truthfully (the "Offer") and then (ii) authorized by the user (the "Acceptance"). We need irrefutable evidence for all stages, including payment and service performance. Currently, we assume that transactions are not intentionally manipulated. In a world with more cybercrime or ASI, both sides of a transaction take risks in every online/business transaction.

There are multiple ways to create Trustworthy eCommerce. Watchdogs and KS/EDU could generate or store transaction evidence. We must have standardized features that would require both sides of a transaction to generate by default irrefutable transaction evidence as part of every eCommerce transaction step. Some components were discussed in [47]. Furthermore, instead of using complex device or interface implementations vulnerable to OS manipulations, we should better use simple, dedicated secure confirmation interfaces (SCI) that are either associated with the IT device or personally held by users. It is conceivable that additional video evidence may be requested or automatically generated to link users/owners to a confirmation step.

6.4 Trustworthy Communication with Legitimate Surveillance

There are good reasons for surveillance of humans: law enforcement and governments demand it because crime-fighting is their service to the community. A similar argument applies to voice, video, or text surveillance between ASI and humans or organizations because there is the risk that ASI could bribe or blackmail people and organizations.

People do not expect privacy in the public sphere, and many cities already have comprehensive surveillance programs for public safety. But surveillance is also a serious intrusion into someone's privacy and civil rights. There are situations in which surveillance or eavesdropping is accepted:

1. Parents have the right to protect their underage children online.
2. Technology for illegal purposes should not create wrong winners. Law enforcement must protect public against criminals.

However, there are issues: governmental overreach is considered a problem even in authoritarian regimes because who controls the controller? Independent oversight via court orders could prevent rogue bureaucrats, criminals, or ASI from using the same interfaces for other nefarious goals.

Most governments in the world demand that eavesdropping on the communication between humans must be possible. Once unbreakable communication is feasible, we must find a common technical foundation from which all countries can build their own systems:

1. E-KS/EDU should receive digitally certified warrants or court orders, allowing KS/ EDU to share session keys while law enforcement eavesdrops/records the encrypted message (outside) on the wire (and parents, e.g., within their Intranet)
2. Warrants or court orders in E-KS/EDU are time- or scope-limited (i.e., which application or websites are covered), and who are the receiver(s) of session keys
3. Warrants/court orders must detect and respect territoriality and ownership of devices
4. The concept of warrants/court orders could also support scripts or trigger criteria for E-KS/EDU to allow, e.g., parents to receive data from their underage teens
5. E-KS/EDU's features/software must be under public and open-source scrutiny

Hashcodes with additional attributes (Enhanced Hashcodes) could enable KS/EDU to make important inferences on what kind of entity is on the other side of the communication. Thereby E-KS/EDU would be able to detect if users are communicating with other humans or with servers or ASI. Hidden metadata on source, type, and/or key purpose associated with the key-related hashcodes can be used by E-KS/EDU to trigger follow-up operations automatically, e.g., helping to acknowledge transactions manually or sending session keys to other servers to facilitate or automate legitimate surveillance. The advantage of including additional data within hashcode becomes clear when we need to change data related to teens when they become adults without changing the underlying private/public keys.

7 Discussion of Near-Perfect vs. Achievable Security

The zero incidence goal in Near-Perfect Security is a motivating goal, like zero accidents in air traffic or nuclear energy. This paper assumes that humans in the operation of security are a liability. Humans should not see keys in cleartext or process them in their main

CPU. Detection of failures should be automated, including investigating possible security incidences to preserve privacy.

Unfortunately, Near-Perfect Security is not pragmatic enough to accept failures as chances to learn and improve security measures and processes. Instead, sufficient key or device security has to share zero tolerance for every possible point of failure, as in aviation accident investigations. Every detected and investigated incident could have much worse consequences. Therefore, using black-box-type recordings of relevant evidence, automatically generated pre-investigation data, and evidence gathering in non-prevented security failures is extremely important for learning, improving, and preparing for future threats. Implementing measures to protect devices or keys from attackers gaining control with methods that are based on past knowledge and experiences is the absolute minimum; without that level of protection, there is no security. Detecting failures automatically, i.e., gathering evidence of cleartext key usage outside EDU, is critical to sufficient security.

7.1 Detecting Stolen Secret Keys

Although we should prevent even the theoretical chance of having encryption keys being leaked in cleartext (covertly), we can never know if we are successful. Even if we assume stealing keys is beyond ASI's abilities, we require ubiquitous overlapping usage detection for successfully stolen keys. Suppose ASI can probe isolated EDU instances; many, potentially all, keys could be extracted from KS, i.e., more than the acceptable worst case. In this super worst-case, attackers could leave probed KS/EDUs intact and users unaware of the attack. Then we would only have left ASI with one uncertainty: has it extracted all keys? Exacerbating this situation for ASI requires KS data records that are unencryptable or genuine-looking decoys, including honeypot keys.

Stolen public keys are not immediately announced as deactivated by KAS. Additionally, private keys for all exposed public keys are considered successfully calculated (e.g., with quantum computation). The compromised system could be a client, a server, or even a security-related service (categorized as a server). Because ASI succeeded in stealing keys, we assume that ASI is running a software version/simulation of the attacked KS/EDU hardware appearing as a client or server. Knowing public/private keys, ASI can do Man-In-The-Middle-Attacks (MITMA) for the attacked but intact EDU. The attack scenarios that we need to consider are, therefore: ASI (a) imitating client instances, (b) pretending to be a server instance, and (c) operating an invisible (potentially manipulative) middleman between the attacked client and server.

In all three scenarios, the task is to detect that the other party is a software instance trying to simulate KS/EDU. The detection goal is to provoke a response that a hardware version would not have done. If a software version's masquerade is about to be detected, we assume that the attacker either makes things up (i.e., lies) or creates a communication interruption (see section 4.6.3). Both responses trigger automated follow-up inspections capable of creating evidence for an attack. Non-compromised EDUs are expected to report unexpected behavior immediately.

The detection of stolen keys can be done during the initialization of the communication, i.e., in the key exchange (see section 4.6.1). Alternatively, EDU must detect data manipulated during data exchange, i.e., both sides creating metadata about their previous communication (see section 4.6.2) and compare them. Detecting a passive use of crypto keys (on eavesdropped encrypted data) would require other tools not considered in this proposal.

7.2 Discussion of KS/EDU's Key Security

Data exchange is safe if symmetric session keys are kept secret and are crypto-analytically unbreakable. After the exchange, session keys are on both sides in protected hardware (i.e., KS/EDU). These session keys (unknown size/type) are safe within KS/EDU if not delivered to legitimate third parties for legal reasons. Also, keys stored in separatable Key-Safe are safe, as they are protected by a unique hardware system that would change its operation when attacked. KS data are not lost because of an EDU reset. A backup file could automatically rebuild KS data from backed-up hashcode references and data received from external TKRs.

Extreme safety within the key and data exchange will cost performance and potentially slow down the initialization of communication. These extreme measures are overkill, useless, and misappropriated scarce resources for most users, organizations, and applications. However, the problem with that argument is that keys must be protected on potentially vulnerable devices against hardware being probed. Top governmental, corporate, or product development activities will likely get more attention from ASI than most users interested in entertainment, private communication, or socializing.

Therefore, based on security clearance, relevance for a corporation or infrastructure, or self-assessment, multiple Levels of Security could be created, e.g., four levels: Top, Important, Support, and Regular. Each level would receive dedicated MESK keys. Regular users do not require MAETS.

Regular devices may not be contacted with higher security keys or receive corresponding hashcode references because they may have low-security key-management features and cannot be trusted. High-security EDUs could use low-security keys, but they would never comingle these keys of different/higher Levels of Security. EDU’s Business rules could prevent public keys from Top/Important clients or servers from being stored on more vulnerable Support-category devices. Top, Important, or Support categories systems could also have more extensive black-box recorders, regularly analyzed by publicly scrutinized software, done by experts with appropriate clearance.

There is no scarcity or prohibitive expensive resource utilization related to keys; managing them is expected to be extremely cheap - fully automated. Compromised keys are deactivated. Old and deactivated keys are treated as possible honeypots by default, but ASI is not expected to use them.

Most importantly, once humanity determines that ASI can undermine EDU’s key security, we could (A) accept that low-security implementation of EDU is insufficient and reduce its distribution over the time accordingly. Alternatively, (B) humanity could conclude that ASI violates rules by breaking humans' security measures and blocking ASI from using certain facilitating services like unrestricted access to quantum computation or manufacturing capabilities. Or (C) humanity could stop and kill ASI globally [38] before it entirely undermines and deactivates humankind’s Kill-ASI capabilities. Suggesting which option should be chosen is beyond the scope of this paper.

7.3 Discussion of KS/EDU's Device-Security

Misusing enhanced EDU devices is a more difficult issue because there are many more details to consider when humans are involved. Basic EDUs within ASI Safety have strict/narrow implementations within allowed usage scenarios and few or no exceptions. Enhanced EDU in eCommerce or human communication has more complicated business rules and potentially many exceptions due to business or political interests. Security should accommodate these features, but predicting if EDU device security could be kept close to near-perfect security cannot be given. Misuse of trustworthy encryption might be likely if the purpose of Enhanced KS/EDU systems is not being kept very narrow.

If EDU is part of a network, i.e., part of a MUS (Multi-Unit-Security), and uses a communication hub, we could use redundancy to create another level of protection against covert EDU manipulation. However, if EDUs are isolated, attackers can probe EDUs' AKE and Key-Safe. Also, attackers could replace isolated EDUs with compromised components trying to deceive the to-be-protected device; the attacker could succeed if the OS is not strict in its refusal.

Even in an IT environment with multiple EDUs, EDUs could be challenged or misused by malware. Without restrictions on software, i.e., without having software binary-hashcoded and whitelisted, it is difficult to predict if EDU cannot be tricked into providing its capabilities for the wrong tool or purpose. Therefore, executable Watchdog-type solutions [39] could facilitate an environment where only known, i.e., binary-hashcoded, executables are accessible by the main CPU.

7.4 Discussion of Peripheral Security Measures

Seeking near-perfect or sufficient security for keys and devices involves controlling multiple peripheral aspects within technical security measures; these features must be proactive and preventative. Manufacturing KS/EDU and their instantiation were discussed as part of the basic implementation (Section 5.4). Another peripheral aspect is EDU's local log data or evidence storage (Section 4.8) and investigation of network or power outages (Section 4.6.3).

We need to be prepared for security breach detection because ASI will not make it easy to gain evidence for ASI's rule violations. When ASI is threatened to be caught, it may start initiating power or connectivity outages as a cover-up. The proposed detection effort remains vulnerable without knowing if interruptions are coincidental or intentional. EDUs could sense the scope of outages and store data in their black box. For causal or statistical conclusions, the outage data should be reported to decentralized network servers.

A beneficial side-effect of minimizing false-positive reporting is that we have an independent tool to help determine the many causes of genuine outages. Due to full transparency, the type and anonymity of reported data do not violate users' anonymity during these automated reportings. If EDU-related security features are implemented right from the beginning, costs of operating automated security are insignificant, and the advantages for our understanding of power, computing, and network infrastructure reliability and trustworthiness could be significant, ahead of ASI.

Additional automation or structures within other Pre- and Post-Security activities, e.g., investigations, false-positive or false-negative reduction in reporting, etc., could enhance the proposed security measures. Still, discussions of these or other enhancements are outside this paper's scope.

8 Conclusions

The main goal of crypto-key security is to defend crypto-keys from being seen in cleartext or used unauthorized. This paper has proposed a technical foundation for significantly improving security against a superior adversary, a hypothetical artificial super intelligence that can modify compiled software and steal unprotected encryption keys.

Zero-incidence rate of Near-Perfect Crypto-Key/Device Security is more of an ideal or motivational goal for establishing sufficient security and proactive or reactive measures to prevent an attacker from gaining unauthorized access to crypto keys. If proposed prevention or protection tools fail, automated detection of stolen keys within data exchange exposes these compromised keys.

An essential feature of Trustworthy Encryption and Decryption is to separate security- and crypto-key-related from regular, versatile computation on CPU, GPU, or unsecured microcontrollers under the control of the main OS via the use of protected key-safes linked to unique EDUs. EDUs, in a basic ASI Safety version, validate and update operating software, receive commands/data, and collect and send evidence to dedicated remote servers as part of an early warning system that works entirely independently of human participation. Shielding crypto-keys from a key-stealing ASI is the new defense line in protecting humanity in a conflict with a rogue ASI that we may have to switch off in our defense against its resistance.

Using isolated crypto devices in hostile environments is an unprotectable situation. Crypto devices must be connected and interguarding their operational integrity. Using secrets from its initialization and continuous use, EDU validates that other EDUs are hardware-based. A software simulation of an EDU is a sure sign of a successful attack.

Standard-compliant, tightly controlled Enhanced EDUs, supporting a narrow feature focus, must be used in communication and eCommerce to establish trust within our technical infrastructure. The generation of irrefutable evidence is required to eliminate or mitigate cyber-crime. Unbreakable encryption must be restricted with legitimate backdoors for providing court-ordered access to session keys that can only be used for passive eavesdropping by law enforcement or concerned parents. If separated from the main CPU/OS, legitimate surveillance can technically be implemented without granting malicious actors the same access.

With the protection of crypto-keys/devices, humanity has the foundational tools to create additional protection solutions against an adversary intellectually and operationally far superior to humans.

REFERENCES

[1] Wikipedia, "Artificial General Intelligence", https://en.wikipedia.org/wiki/Artificial_general_intelligence, Last Visited 14/05/2021

[2] I.J. Good, "Speculations concerning the first ultraintelligent machine", Advances in Computers, Vol. 6, 1966, Pages 31-88, doi:10.1016/S0065-2458(08)60418-0

[3] N. Bostrom, "Superintelligence: Paths, Dangers, Strategies" Oxford UP, (2014)

[4] lesswrong.com, "Intelligence Explosion" https://www.lesswrong.com/tag/intelligence-explosion, Last Visited 14/05/2021

[5] CNBC, 6 Apr 2018, “Elon Musk warns A.I. could create an ‘immortal dictator from which we can never escape’”, https://www.cnbc.com/2018/04/06/elon-musk-warns-ai-could-create-immortal-dictator-in-documentary.html, Last Visited 13/05/2021

[6] BBC News, 29 Jan 2015, “Microsoft's Bill Gates insists AI is a threat” https://www.bbc.com/news/31047780 Last Visited 13/05/2021

[7] BBC News, 2 Dec 2014, “Stephen Hawking warns artificial intelligence could end mankind” https://www.bbc.com/news/technology-30290540, Last Visited 13/05/2021

[8] Claude Shannon, 1949, “Communication Theory of Secrecy Systems” in Bell System Technical Journal 28(4) pp. 656–715

[9] Niels Ferguson, Bruce Schneier, Tadayoshi Kohno, “Cryptography engineering design principles and practical applications” Wiley Publishing, Inc. (2010)

[10] Ross J. Anderson, “Security engineering: A guide to building dependable distributed systems”, 3rd Ed., Wiley (2020)

[11] M. Alfonseca, M. Cebrian, A.F. Anta, L. Coviello, A. Abeliuk, I. Rahwan: “Superintelligence cannot be contained: Lessons from computability theory”, Journal of Artificial Intelligence Research 70 (2021) 65-76, doi:10.1613/jair.1.12202, arXiv:1607.00913 [cs.CY]

[12] R. V. Yampolskiy, "On Controllability of AI" (2020), arXiv:2008.04071

[13] Stefan Heule: “How Many x86-64 Instruction Are There Anyway” (2016) https://stefanheule.com/blog/how-many-x86-64-instructions-are-there-anyway/, Last Visited 13/05/2021

[14] Fabian Giessen: “How many x86 instructions are there?” (2016), https://fgiesen.wordpress.com/2016/08/25/how-many-x86-instructions-are-there/, Last Visited 13/05/2021

[15] Larry Seltzer: "Securing your private keys as best practice for code signing certificates" https://www.thawte.com/code-signing/whitepaper/best-practices-for-code-signing-certificates.pdf Last Visited: 14/05/2021

[16] Barker, Elaine (May 2020). "NIST Special Publication 800-57 Part 1 Rev 5, Recommendation for Key Management: General" (PDF). National Institute of Standards and Technology. doi.org/10.6028/NIST.SP.800-57pt1r5. Retrieved 2022-05-20

[17] Elaine Barker, William C. Barker (May 2019). "NIST Special Publication 800-57 Part 2 Rev 1, Best Practices for Key Management Organizations" (PDF). National Institute of Standards and Technology. doi.org/10.6028/NIST.SP.800-57pt2r1. Retrieved 2022-05-20

[18] Elaine Barker, Quynh Dang (Jan 2015). "NIST Special Publication 800-57 Part 3 Rev 1, Recommendation for Key Management: Part 3: Application-Specific Key Management Guidance" (PDF). National Institute of Standards and Technology. dx.doi.org/10.6028/NIST.SP.800-57pt3r1. Retrieved 2022-05-20

[19] B. Dang, A. Gazet, E. Bachaalany: “Practical reverse engineering x86, x64, ARM, Windows kernel, reversing tools, and obfuscation” (2014), John Wiley & Sons, Inc.

[20] Waratek "Runtime Application Self Protection (RASP) evaluation criteria" Mar 2016, https://ten-inc.com/documents/RASP_Evaluation_Criteria.pdf Last Visited: 14/05/2021

[21] C. Paduraru, M. Paduraru, A. Stefanescu: "Optimizing decision making in concolic execution using reinforcement learning", 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), March 2020, DOI: 10.1109/ICSTW50294.2020.00025

[22] IBM CEX6S (4768) PCIe Cryptographic Coprocessor (HSM), 2019, https://www.ibm.com/downloads/cas/JMD7BBN4, Last Visited: 14/05/2021

[23] Rivest, R. L., Adleman, L., & Dertouzos, M. L. (1978). On data banks and privacy homomorphisms. Foundations of secure computation, 4(11), 169–180.

[24] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Ange-la Jäschke, Christian A. Reuter, Martin Strand, "A Guide to Fully Homomorphic Encryption" (2015), https://eprint.iacr.org/2015/1192, Last Visited: 17/06/2021

[25] Melissa Chase, et al. Security of Homomorphic Encryption (Homomorphic Encryption Standardization Workshop on July 13-14, 2017) http://homomorphicencryption.org/white_papers/security_homomorphic_encryption_white_paper.pdf, Last Visited: 17/06/2021

[26] Wikipedia, “Key size”, https://en.wikipedia.org/wiki/Key_size, Last Visited 2022-05-20

[27] Michael Cobb, Oct 2013, "The value of 2,048-bit encryption: Why encryption key length matters", https://www.techtarget.com/searchsecurity/answer/From-1024-to-2048-bit-The-security-effect-of-encryption-key-length, Retrieved 2022-05-24

[28] Neil Coffey, 2021, "RSA key length", Javamex: Java Tutorial, https://www.javamex.com/tutorials/cryptography/rsa_key_length.shtml, Retrieved 2022-05-24

[29] M. Sabt, M. Achemlal, A. Bouabdallah: "Trusted Execution Environment: what it is, and what it is not", 2015 IEEE Trustcom/BigDataSE/ISPA (2015), 57-64, DOI: 10.1109/Trustcom.2015.357

[30] Winter, J. Trusted computing building blocks for embedded linux-based ARM trustzone platforms. In Proceedings of the 3rd, ACM workshop on Scalable trusted computing, New York, NY, USA, 31 October 2008.

[31] Raj, H.; Saroiu, S.; Wolman, A.; Aigner, R.; Cox, J.; England, P.; Fenner, P.; Kinshumann, C.; Kinshumann, K.; Loeser, J.; et al. TPM: A Software-Only Implementation of a TPM Chip. In Proceedings of the 25th USENIX Security Symposium USENIX Security 16, Austin, TX, USA, 10–12 August 2016.

[32] The Wall Street Journal, WSJ News 12. Feb 2020: "U.S. Officials say Huawei can covertly access telecom networks" https://www.wsj.com/articles/u-s-officials-say-huawei-can-covertly-access-telecom-networks-11581452256, Last Visited 14/05/2021

[33] Glenn Greenwald: "No Place to Hide: Edward Snowden, the NSA, and the U.S. Surveillance State", Metropolitan Books, (2015)

[34] ExtremeTech 16 Sep 2013: "Researchers find new, ultra-low-level method of hacking CPUs – and there’s no way to detect it" https://www.extremetech.com/ extreme/166580-researchers-find-new-ultra-low-level-method-of-hacking-cpus-and-theres-no-way-to-detect-it Last Visited 14/05/2021

[35] Jack Wallen, 1 Jul 2016: "Is the Intel Management Engine a backdoor?", https://www.techrepublic.com/article/is-the-intel-management-engine-a-backdoor/ Last Visited: 14/05/2021

[36] ZDNet, 7. Nov 2018 "Cisco removed its seventh backdoor account this year, and that's a good thing" https://www.zdnet.com/article/cisco-removed-its-seventh-backdoor-account-this-year-and-thats-a-good-thing/, Visited 14/05/2021

[37] Darren Moffat, "Is FIPS 140-2 Actively harmful to software?" 16 Apr 2014, https://blogs.oracle.com/solaris/is-fips-140-2-actively-harmful-to-software-v2 Last Visited 14/05/2021

[38] E. Wittkotter, R. V. Yampolskiy: “ASI Safety via Technologies to Switch-Off/Kill ASI”, unpublished

[39] E. Wittkotter, R.V. Yampolskiy: “No-Go for malware using independent executable watchdog" unpublished.

[40] "Perfect" Merriam-Webster.com. 2011. https://www.merriam-webster.com (8 May 2011).

[41] Wikipedia, “Physical unclonable function”, https://en.wikipedia.org/wiki/Physical_unclonable_function, Last Visited 09/07/2021

[42] Ulrich Rührmair (2012), "SIMPL Systems as a Keyless Cryptographic and Security Primitive", in Cryptography and Security: From Theory to Applications, Jan 2012, DOI:10.1007/978-3-642-28368-0_22, Retrieved 2022-05-22

[43] Ulrich Rührmair, 2022, "Secret-free security: a survey and tutorial", Journal of Cryptographic Engineering, https://doi.org/10.1007/s13389-021-00283-6

[44] Kerckhoffs, A.: La cryptographie militaire. J. Sci. Mil. (1883)

[45] WebFx, 13 Mar 2020: "What is the Real Cost of Computer Viruses? [Info-graphic]" https://www.webfx.com/blog/internet/cost-of-computer-viruses-infographic/ Last Visited: 14/05/2021

[46] Steve Morgan, 13 Nov 2020: "Cybercrime to cost the world $10.5 trillion annually by 2025 https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/ Last Visited: 14/05/2021

[47] E. Wittkotter, “Trustworthy encryption and communication in an IT ecosystem with artificial superintelligence", in Proceedings of 5th Workshop on Attacks and Solutions in Hardware Security (ASHES '21), doi.org/10.1145/3474376.3487279