IT Security for Developers

A Drama in Three Acts

Security is perhaps the most neglected area of information technology. It is extensive, diverse and complex. It is political. It is ungrateful. It is a drama. An attempt to cope in three acts.

Version 0.4.2, June 2022

http://dck.one, txt@dck.one

Prologue

The Stage

A small company but *not* a corporation is the setting.

The Scenes

The security concept of the BlockKeeper application is explained in three acts: The first act illuminates the basics and the context. Theoretical conceptualization and practical implementation follow in the second and third acts.

The Protagonist

An experienced developer who is not a security expert, having an inner dialogue.

The Author

Is working over 25 years in IT and mechanical engineering businesses as architect, developer, UNIX system/network administrator and entrepreneur. He was one of the lead BlockKeeper programmers in 2017.

As cofounder of Micro-Colocation.com, his focus since 2021 has been on green edge computing for small computers like Raspberry Pi, Odroid and Jetson Nano. CC licensed texts like this drama are created within the working hours of this company. You can support the text production by sharing the Micro-Colocation.com link in your network and, of course by booking colocation services. Many thanks.

Table of Contents

First Act

First Scene – The Dilemmas

There are at least three basic dilemmas in information technology:

The Economic Dilemma

To achieve the desired profit in capitalistic enterprises, minimization of production expenditure is a decisive factor. As in other industries, the costs correlate closely to production time. The maximum amount of time is therefore usually hard-coded and leads to the following problem: If it is exceeded, the development speed is too slow and the product is not viable. Either the competition has already served the market or the costs are beyond the acceptable range. If it is not exceeded, there is never enough time to implement a *perfect* product in all aspects.

The Need Dilemma

The need for innovative, functional, intuitive, beautiful and glittering products is high. The need for secure products is low. Security only becomes important when it is too late.

The Security Dilemma

This is the exemplary design philosophy of the security and privacy-focused email service ProtonMail:

> *Our basic assumption is that all servers can be compromised, and that sooner or later, ProtonMail will also be compromised. Incidents like the Yahoo breach are not isolated, and will gradually become the norm over the next decade.*
>
> ProtonMail Blog: Improved Authentication for Email Encryption and Security

One aspect, in particular, makes the topic of security difficult to master: weak points often do not have local (feature) but global (product) effects. Even the smallest errors in design or implementation can endanger the security of the entire system. Significant effort does not necessarily result in a secure product. The Web, as the information technology platform par excellence, must also contend with the Web dilemma:

> *Cryptography is a systems problem, and the web is not a secure platform for application delivery. The web is a way to easily run untrusted code fetched from remote servers on-the-fly. Building security software inside of web browsers only makes the problem harder.*
>
> Tony Arcieri: What's wrong with in-browser cryptography?

Security is complex.

The Consequences

Security is just one of many product aspects and often has a low priority due to the need dilemma.

Due to the low priority and the time constraint imposed by the economic dilemma, the scope of security mechanisms must be sensibly chosen: too many measures result in a partial implementation, whereas too few measures lead to an unusable product.

The starting point for a meaningful selection of measures is a risk assessment (risk rating) that answers the key question: What security precautions must be taken to ensure that the (economic) damage for the user and producer remains acceptable in the event of a fault?

Even after a sound risk assessment, consistently finding consensus in terms of team, corporate, internet, concept is difficult due to the security dilemma. Criticism of the selection of measures should therefore be understood as process-inherent, accepted and used for continuous improvement.

A security-optimized IT product requires sound design, a complete implementation and continuous improvement of a *content-wise and economically suitable* concept.

Second Scene – The Developer

The protagonist of the drama is an experienced developer. Not an IT security expert. Not a cryptographer. This difference is essential: developers are accustomed to familiarizing themselves with abstract problems, but IT security has a special position for two reasons:

1. The security dilemma exists: System security is extensive and complex and the economic dilemma allows only little training time. It takes a lot of basic knowledge to approach the topic in a meaningful way.
2. Security is hard to verify: predicting whether a chosen implementation will withstand any future attack is not possible.

While the developer will say "but the modified data will come back as garbage after decryption", a good security engineer will find the probability that the garbage causes adverse behaviour in the software, and then he will turn that analysis into a real attack.

Stack Overflow: Post by user TheGreatContini

Smaller companies, and especially startups, that produce innovative solutions with comparatively low security standards usually do not have security experts in their own ranks. Ignoring the security dilemma cannot be a solution. Instead, coping with this falls within the scope of the in-house developers. And they need: allies.

Third scene – The Allies

Literature and Standards

Not surprisingly, the closest allies for familiarizing yourself with a subject are specialized publications and standards. While studying them, the challenge lies in the effective filtering of information: both the selection and the cross-reading of the literature must be focused on the *implementation*. It helps to be clear during the training: A deep study of the hard (mathematical cryptography) theory is not possible due to the economic dilemma. It is also not necessary because the software engineer does not conduct basic research, but must dominate the application.

The standardization of algorithms and processes is an elementary component of the IT security world. Although of greater importance as a source of information for the researcher, official guidelines such as those published by NIST (SP 800, Computer security) and RFC are also of interest to developers for answering detailed questions. Since not every code library includes extensive documentation, it is often unclear what individual parameters mean. A look at the corresponding NIST SP 800 algorithm description can be helpful in such a case.

Federal Authorities

Due to the great importance of information technology, many countries have authorities focused on it. Their responsibilities include specifying cryptographic standards, documenting, warning of security vulnerabilities, and reviewing as well as licensing IT systems for security purposes.

The National Institute of Standards and Technology (NIST) is particularly well known, in Germany also the Bundesamt für Sicherheit in der Inforamtionstechnik (BSI). The BSI's technical guidelines and comprehensive IT security compendium (IT Grundschutz Kompendium) offer not only a general introduction, but also specific recommendations on the subject of IT security. For developers, it is more than helpful to

receive, for example, concrete information on cryptographic key lengths and a catalog of important security aspects for small businesses.

However, experts like the members from the Chaos Computer Club discuss lively whether the focus of federal authorities is on securing (critical) systems or on the concerns of intelligence services, law enforcement agencies or even military interests.

Neither ignoring nor overstating this open discussion, authority guidelines play no further role in the following text. Reasons are the required sources number limitation due to the targeted text length as well as the underlying open content idea of this drama. It is better transported via the use of international, public community posts and articles.

Best Practices

Thanks to the now widespread open source culture, extensive documentation, concepts and best practices can be found on every IT topic, such as Lessons learned and misconceptions regarding encryption and cryptology. Even at an early stage, it is important to identify the appropriate methods for your own product development. If your own product, for example, has to perform an extreme balancing act between usability and security (need dilemma), it may be worthwhile to look at the Signal Messenger team's approach, as it has to contend with the same requirements.

I am regularly impressed with the thought and care put into both the security and the usability of this app. It's my first choice for an encrypted conversation.

Signal Hompage: Bruce Schneier about the Signal Messenger

Also, the password manager software LastPass and many other projects give insight into their technologies. Thorough research and careful selection in this environment can significantly shorten the design of the concept, thereby allowing more time for implementation.

And the de facto king of principles must not go unmentioned. Especially in the area of security and cryptography, only this regiment can prevent the downfall of the kingdom:

Keep it simple, stupid: The KISS principle states that most systems work best if they are kept simple rather than made complicated; therefore simplicity should be a key goal in design and unnecessary complexity should be avoided.

Wikipedia: The KISS Principle

Code Bibliotheken

Have to be used! The Don't Roll Your Own Crypto mantra must be the supreme rule for any developer: Software developers are *users* of security and cryptography solutions.

I learned how easy it is to fall into a false sense of security when devising an encryption algorithm. Most people don't realize how fiendishly difficult it is to devise an encryption algorithm that can withstand a prolonged and determined attack by a resourceful opponent.

Phil Zimmermann: An Introduction to Cryptography

So, developers have to use appropriate, security-tested, ideally standardized code libraries to implement security features.

Security is only as strong as the weakest link, and the mathematics of cryptography is almost never the weakest link. The fundamentals of cryptography are important, but far more important is how those fundamentals are implemented and used.

Bruce Schneier: Practical Cryptography Preface

Experts

At the beginning it is even for experienced software developers not easy to find the right path in the jungle of IT security. The selection of literature is difficult, since the own information filters are not yet

calibrated. However, as in any community, there are many professionals in the IT security industry who enjoy a special status of trust because of the work they have done. Their publications and lectures can be a starting point for further research. For example, you often find the names Bruce Schneier, Matthew Green, Phil Zimmermann and D. J. Bernstein. Also native speaking experts are of course available in every country. Not only in Germany, Prof. Rüdiger Weis is a known cryptographer who in many Chaos Computer Club talks points out the importance of the correct security library usage. And all of the mentioned are just the tip of the iceberg, many more experts, who often support beginners in their free time, can be found in online communities.

Yet it may be roundly asserted that human ingenuity cannot concoct a cipher which human ingenuity cannot resolve.

Edgar Allan Poe

The wisdom of not blindly trusting the opinion of individuals also applies here and is forced by the security dilemma. Unclear concept points should therefore not only be discussed with the own team (and associated individual experts), but in case of doubt also with a larger number of experts…

Communities

On the subject of security, there are numerous sources of information and communities on the internet. Below, two of them are mentioned as examples that played an important role in the implementation that will be described in the second and third acts.

OWASP: The OWASP Foundation is a well-known community in the IT security environment, which, amongst others, publishes action recommendations. Publications, such as The Ten Most Critical (Web) Application Security Risks, are particularly helpful for the initial risk assessment, which is the basis for any further conceptual decisions.

OWASP is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted. All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security.

About the OWASP Foundation

Stack Exchange: Both in the concept phase as well as during the later implementation, specific and detailed questions are regularly asked which cannot be answered by deep research. *The* Q&A community network for such cases is known to every developer: Stack Exchange. Help in the area of IT security can be found especially in the following exchanges:

Information Security Stack Exchange is a question and answer site for information security professionals.

About the Information Security Stack Exchange

Cryptography Stack Exchange is a question and answer site for software developers, mathematicians and others interested in cryptography.

About the Cryptography Stack Exchange

In both communities, the outrageous person usually receives a professional answer within a few hours and can, if there is any doubt, ask for more details about the comment dialog. Prerequisite, however, is compliance with the Stack Exchange ethos:

Focus on questions about an actual problem you have faced. Include details about what you have tried and exactly what you are trying to do. Not all questions work well in our format. Avoid questions that are primarily opinion-based, or that are likely to generate discussion rather than answers.

The Stack Exchange Rules

What should my security concept look like? will therefore lead to few or no responses. Instead, there must be specific questions like *What does parameter x mean in the case of encryption algorithm y in the context of z?* The basis for this is a concept developed with the help of the other allies.

Fourth Scene – The Concept

Although many product details are not known at the beginning of a project, it is essential to create a first rough concept as the basis for the subsequent process steps.

The Draft

Part of the draft is an estimate of the desired security standard, which results from an initial weighting of the dilemmas. In combination with other product and process parameters, it is then possible to answer the technology questions inherent in the IT environment. For example, the choice of platform (web, mobile, desktop application) is crucial. It sets the framework for the programming languages, code frameworks, and libraries.

The Risk Evaluation

In terms of the design, it is important to classify the individual aspects of the dilemmas in more detail with regard to the planned product. Assessments of this kind are a tiny component of risk management (or information security management). Its complete textbook implementation (and certification) takes on gigantic proportions and cannot be done by a small business. Instead, the developer should focus on a highly simplified risk rating: 1.) To what extent is the new product affected by dilemma or aspect? 2.) What effects does it have on the business and development process? 3.) What is the priority of the measures resulting?

However, answering these few key questions remains difficult. Help can be found, for example, at the OWASP Foundation, which publishes the Risk Rating Methodology:

Discovering vulnerabilities is important, but being able to estimate the associated risk to the business is just as important. […] By following the approach here, it is possible to estimate the severity of all of these risks to the business and make an informed decision about what to do about those risks. Having a system in place for rating risks will save time and eliminate

arguing about priorities. This system will help to ensure that the business doesn't get distracted by minor risks while ignoring more serious risks that are less well understood. Ideally there would be a universal risk rating system that would accurately estimate all risks for all organizations. But a vulnerability that is critical to one organization may not be very important to another. So a basic framework is presented here that should be customized for the particular organization.

OWASP: Risk Rating Methodology

Specifically, OWASP regularly publishes a summary of the Ten Most Critical (Web) Application Security Risks that can serve as a practical guide to risk assessment.

The Planning

During the conceptualization, the requirements identified in the risk assessment are linked with considerations of available technologies and finally translated into concrete implementation tasks.

To produce a secure web application, you must define what secure means for that application.

OWASP: Top 10 2017 Web Application Security Risks

In doing so, the allies help make the right decisions: while initially focusing on literacy and the study of action recommendations, more and more specific code examples and the help of Q & A communities are increasingly important over time. Particularly in the case of agile software development, the conceptual design, prototyping and implementation phases are permanently mixed, since the selection of suitable security solutions depends on many factors and requires associated (technology) tests. Also, the issue of the availability of suitable code libraries is a crucial factor, in reality. It influences the early concept phase and is characterized by the product platform as well as the programming language used. For example, current browsers have a standardized cryptographic interface that defuses the web dilemma, at least in this area. In practice, this development makes it possible to

consider whether a product originally planned as a mobile app could perhaps also be realized as a web service.

The curtain falls.

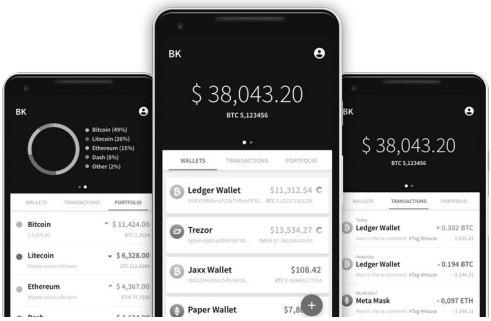
Second Act

In the second act, we follow up on the previous exposition with a description of the security concept developed for BlockKeeper. Special attention is paid to the role of the allies and the compromises that had to be made during planning due to the dilemmas. In order not to strain the scope of this performance, the following scenes focus on three main topics: authentication, authorization and encryption. The main allies are: The Top 10 Web Application Security Risks, published by the OWASP Foundation as well as Lessons learned and misconceptions regarding encryption and cryptology from the Information Security Stack Exchange blog.

Please note: The first short draft of this drama performance was created in late 2017. The author of this performance was one of the lead developers of BlockKeeper at the time. BlockKeeper itself has not been developed further since 2019. However, its approaches and technologies were idea generators for later projects.

First Scene – The Application

To evaluate the conceptual decisions that were made, it is first of all important to precisely define the properties, goals and environment of the application.



Track your crypto-portfolio, secure and anonymous.
The world is going to run on Blockchains and the number of digital coins and assets are exploding. BlockKeeper is your book keeping App for Blockchain assets that helps you to track all your digital wealth, income and expenses.

BlockKeeper Homepage 2017

The Properties

BlockKeeper provides users with information about their cryptocurrency assets, managing a form of financial data that is fundamentally sensitive. However, it only has read-only access to this data. Specifically, BlockKeeper uses and stores only public blockchain addresses (public keys) to represent balances and transaction values of cryptocurrencies (coins).

Individual public addresses are not a secret in themselves: their balances can be viewed at any time via the so-called block explorer (e.g., Blockchain.info). To be able to spend the coins, the associated private key is always required. Since BlockKeeper does not have this key, even the largest data leak cannot lead to a loss of assets for the user.

However, which address belongs to which user is not publicly visible: Blockchains only know addresses, transactions and associated balances, they do not store information indicating that public key x is assigned to person y. Common cryptocurrencies therefore have some degree of pseudo-anonymity.

An application that provides a balance sheet must necessarily establish the relationship between a user and their public addresses within its processes. Since BlockKeeper does not store any personal information, such as real names or email addresses, this part of the pseudo-anonymity is preserved. Nevertheless, the stored relationship information (hereafter referred to as coin addresses) are sensitive and must be protected as much as possible from the eyes of third parties.

The Platform

The Web is an excellent place to display information. Easy to use, popular and largely independent of the operating system are good arguments if the platform question arises for a product. The benefits of the Web help to reduce the need dilemma, but at the same time, the browser interface also aggravates the web dilemma:

> *Whether you are new to web application security or are already very familiar with these risks, the task of producing a secure web application or fixing an existing one can be difficult. If you have to manage a large application portfolio, this task can be daunting.*
>
> OWASP: Top 10 2017 Web Application Security Risks

Nevertheless, the BlockKeeper team chose the Web as the platform during the design phase 2017 because it offered the best balance between requirements and dilemmas *for this product*. Not so long ago, this decision would have been different: browsers had just recently introduced the WebCrypto-API, which provides standardized access to cryptographic JavaScript functions. Although this interface has been criticized by many experts and it certainly does not solve the fundamental web dilemma, this ally was still crucial to achieve the security standards defined in the design phase. The consequence of this choice is also that BlockKeeper only supports newer generation browsers and thus aggravates the need dilemma.

The User Experience

An important factor, which is part of the need dilemma, is the user experience (UX) of an application. Many web applications are sub-optimal in this regard, also because frequent load times interfere with usage. With the help of modern browser functions and JavaScript frameworks, however, it is possible to provide users with fluid navigation, like users of desktop or mobile applications are already familiar with. An important feature of BlockKeeper in this context is local storage: Its use as primary storage for all user data significantly reduces communication

with the backend and thus load times, but has implications for the security dilemma.

The Third Parties

With security and privacy definitions, so-called third parties quickly enter the discussion. In the conceptualization phase, the central question arises:

Who are the third parties (attackers, enemies, …) against whom the user data must be protected?

In the case of BlockKeeper, it is more specific: Which person is allowed to access the sensitive coin addresses? The answer is: only the user. The answer to the same question becomes more difficult when viewed from the perspective of the application:

Which system components may access the user data?

With perceived 90 percent of the applications currently available on the IT market (regardless of industry, function or other content-related criteria), the answer is: All. This means that data is never processed only on the device of the user, but always transferred to the backend of the operator and stored there. As a result, not only the user, but also, the service operator has access to the data.

For many services, this measure is comprehensible, useful or even mandatory. For a product like BlockKeeper, however, we at the BlockKeeper team took the view that neither malicious attackers nor operators should have access to user data. Accordingly, the security concept was designed.

Second Scene – The Identification

The security level of authentication and authorization depends, in particular, on one factor: the credentials (username / password). So, here the developer immediately finds himself in the need dilemma:

People are notoriously poor at achieving sufficient entropy to produce satisfactory passwords.

Wikpedia: Human-generated passwords

The User Identifier

BlockKeeper puts log-in convenience in the background in favor of security: the user cannot choose a user name when registering as is typical, but instead, the user is assigned a so-called user-identifier by the system. The downside is obvious: it's impossible for people to remember a string like 74ad7a9a-3c0b-4dad-8aa9-c80437506605. In the case of BlockKeeper, however, this factor loses significance for several reasons.

- The user is logged in directly when registering and normally does not need to log in again (on the same device): The persistent storage of the necessary data in the local storage of the browser also survives restarts of the browser and the operating system.
- The cryptocurrency community is accustomed to coming in contact with cryptographic keys. The savekeeping of even complex credentials is therefore rather familiar to members as opposed to unfamiliar.
- The acceptance of the credentials by the user is facilitated by the support of password managers, a suitable design of the registration mask, a simple backup function and associated FAQ support.

The loss of comfort is offset by several advantages:

- When the user identifier is generated using a cryptographically secure random number generator (CSPRNG), it will contain sufficient entropy (Make sure you seed random number generators with enough entropy) to serve as a secure authentication token. Additional key stretching procedures are not required.

- In this case, also no username/password combination is required; the CSPRNG-generated user identifier combines them into *one* token: due to its randomness, it ensures the unique identification of the user (username) and at the same time ensures secure access control (password). The token must be evaluated as appropriately sensitive and must necessarily be protected from the eyes of third parties.
- Since many users cannot remember a fictitious username, they tend to use their email address, which is contrary to the basic BlockKeeper concept: no personal data is stored. By setting a user identifier, this predicament is avoided from the start.

The Crypto Key

The information managed by BlockKeeper is, by definition, only accessible to the user, but must still be stored reliably (i.e., not just on the user device). BlockKeeper solves this need dilemma by combining server-side storage with client-side encryption:

Client-side encryption is the cryptographic technique of encrypting data on the sender's side, before it is transmitted to a server [...] Client-side encryption features an encryption key that is not available to the service provider, making it difficult or impossible for service providers to decrypt hosted data.

Wikipedia: Client-side encryption

The key required for the encryption has the same entropy requirements as the user identifier in terms of its generation. However, since the latter must be known to the backend, it cannot be used to encrypt the data. The basic rule applies:

Don't use the same key for both encryption and authentication.

Stack Overflow: Post from user roryalsop

In addition to the user identifier, the user thus requires a second, in this case only known to him or her, credential. In the case of BlockKeeper, it bears the name crypto key internally and is generated during registration in the same way as the user identifier. The requirement dilemma is not increased by this: Whether the credential backup or the password manager contains one or two complicated keys is irrelevant.

The Storage

The credentials are constantly required within the processes for authentication, authorization, encryption and decryption. They must therefore be kept in the browser after login and, in the case of the user identifier, sent regularly to the backend. However, closing browser tabs, or the entire browser, is a common process that would result in a renewed credentials query every time. To solve this need dilemma and to ensure a simple transmission of the user identifier, the following basic decision was made for BlockKeeper:

The password manager, the local storage and the HTTPS headers are *sufficiently inaccessible* areas to store or transmit unencrypted information. This means that BlockKeeper can store all data (credentials, coin addresses, etc.) persistently and unencrypted in the local storage, as well as transfer it to the backend in HTTPS header fields.

The Compromise

This decision is a typical compromise, as it is repeatedly necessary because of the dilemmas. Because in terms of application security, this type of storage is in no way perfect. Amongst other drawbacks, it assumes that an attacker is unable to gain physical access to the user's device. If he could, he could read out the sensitive values in the local storage and the HTTPS header using browser web developer tools. So, the compromise decision creates a BlockKeeper-appropriate protection that meets the defined security requirements without ignoring the KISS ally. For another application with different requirements, this compromise would be completely inadequate:

To produce a secure web application, you must define what secure means for that application.

OWASP: Top 10 2017 Web Application Security Risks

Security is a dilemma. Sometimes even a drama.

Third Scene – The Access

A look at the reality, in the unfounded hope of perhaps doing a little better:

Broken Authentications: Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

Broken Access Control: Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

OWASP: Top 10 2017 Web Application Security Risks

BlockKeeper's access rights are very simple: each user can access their resources but not any other resources. To address the resources, during the generation of UUIDs, BlockKeeper exclusively uses a CSPRNG-capable code library. This measure results in the pragmatic approach of being able to realize the user identifier as a UUID and allowing it to assume a dual role:

1. It acts as a normal ID, using standard UUID-based user resource addressing techniques.
2. It represents the entry point to the resources of the associated user. This means that the user can access all resources that are hierarchically subordinate to the user identifier resource.

Authentication and Authorization

This makes it possible to map authentication and authorization in a simple procedure:

- During registration, a user identifier in the form of a UUID is automatically generated for the user in the frontend. Although this generation could be bypassed by an attacker with direct access to the backend API and thus any string could be passed instead. However, this "attack" is prevented by a backend UUID validation and would only lead to the weakening of the attacker's own identifier, but not other user identifiers.
- During the login process, the backend checks whether the user-identifier resource exists. If so, the user is known to the system (identification / authentication).
- Similar to the login, the user identifier is also sent with every further backend request (in the HTTPS header). By matching it with the storage hierarchy, the system can decide at any time whether access to the requested resource is allowed (authorization).

The CSPRNG Factor

Crucial for this conceptual approach is the use of a cryptographically secure random number generator, because the general rule is:

> *Do not assume that UUIDs are hard to guess; they should not be used as security capabilities (identifiers whose mere possession grants access), for example.*
>
> RFC 4122: Chap. 6

The documentation of the code library used must be checked for that. Only if the entropy of the UUIDs is high enough is the user identifier sufficiently protected against collisions and automated derivations.

Fourth Scene – The Encryption

Sensitive Data Exposures: Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

OWASP: Top 10 2017 Web Application Security Risks

BlockKeeper stores all data unencrypted in the local storage, but ciphers it completely before the transfer to the backend. For applications of this type, symmetric encryption, and specifically the application of the Advanced Encryption Standard (AES), has become established as the de facto standard. AES is a popular block-based encryption method. However, encryption is just one side of the coin…

The Integrity

Understanding the difference between data integrity and data encryption is of fundamental importance:

[Symmetric encryption] schemes are not insecure because they leak plaintext information to someone who just intercepts a ciphertext. In fact, most modern schemes hold up amazingly well under that scenario, assuming you choose your keys properly […] The problem occurs when you use encryption in online applications, where an adversary can intercept, tamper with, and submit ciphertexts to the receiver. If the attacker can launch such attacks, many implementations can fail catastrophically, allowing the attacker to completely decrypt messages.

Matthew D. Green: How to choose an AE mode

When we think of encryption, the first thing that generally comes to mind is confidentiality, or keeping data secret. However, there's a second property that's equally important that is integrity, or ensuring that the data we decrypt is the same as the data we encrypted. The way that we do this is

through the use of authentication, in the form of a message authentication code (MAC).

Xanderland: That Crypto Code Sample Is Probably Wrong

The Operation Mode

However, the fundamental rule Don't use encryption without message authentication has, in practice, the difficulty of complexity:

Getting this right is hard all by itself, and if you do it incorrectly, you can leave your implementation exposed to whole classes of attacks, such as timing attacks on the MAC verification process.

Xanderland: That Crypto Code Sample Is Probably Wrong

Fortunately, there is a way out of this part of the security dilemma: thanks to numerous hard-working security experts, there are so-called operation modes that combine encryption and MAC methods reliably and make them relatively easy to use. Therefore, when working on ciphering and the AES algorithm, the developer should not aggravate the economics dilemma by studying the detailed (mathematical) operation, but instead focus on the appropriate operation mode.

In general, the decision of which cipher mode to use is not something most people make every day, but when you do make that decision, you need to make the right one.

Matthew D. Green: How to choose an AE mode

The curtain falls.

Third Act

After the basics and the conceptual design, what remains for the third act: The implementation. It is less wordy and significant, because the audience gets tired, the economic dilemma is breathing down the author neck and this drama is about IT security and not the application BlockKeeper. A quick look at the topic of encryption must therefore be sufficient…

Last Scene: The Implementation

A quick overview of the BlockKeeper architecture:

- The frontend (React) is implemented as a Single Page Application (SPA), which contains most of the application logic.
- The backend (Node.js, today the author would use Python) stores the encrypted user data.
- The communication with the JSON API on the backend, and other external services, takes place via Ajax requests.
- User data is exchanged only in encrypted form between the frontend and the backend, but cached unencrypted in the browser's local storage.

Encryption and Integrity

As described, data integrity and encryption are two sides of the same coin. That‘s why MAC and encryption methods are linked together in operation modes. Developers access them by code libraries, thus avoiding rudimentary failures when combining the conepts.

Authenticated encryption (AE) provides confidentiality and data authenticity simultaneously. An AEAD scheme is usually more complicated than confidentiality-only or authenticity-only schemes. However, it is easier

to use, because it usually needs only a single key, and is more robust, because there is less freedom for the user to do something wrong.

Stack Exchange: Post from user Dmitry Khovratovich

AEAD modes not only provide the integrity and encryption needed, but also hide their complexity under a simplified interface.

The AES Galois Counter Mode

When selecting the AEAD mode, this pragmatic question arises first: which modes does the intended code library support? BlockKeeper uses the WebCrypto API, which in December 2017 contained only one AEAD mode, that was available in all current browsers: AES-GCM.

Galois Counter Mode has quietly become the most popular AE(AD) mode in the field today, [...] if you have someone else's implementation — say OpenSSL's — it's a perfectly lovely mode. [...] Having read back through the post, I'm pretty sure that the 'right' answer for most people is to use GCM mode and rely on a free implementation.

Matthew D. Green: How to choose an AE mode

So, the choice was very simple in the case of BlockKeeper. Should AES-GCM for some reason not be suitable or not available, NIST SP 800-38D gives a overview of other AEAD procedures.

Depending on the application, AES-GCM expects a different number of parameters, which are not very well documented in the WebCrypto API functions in December 2017. But the allied standards come to the rescue, for example, RFC 5084:

AES-GCM has four inputs: an AES key, an initialization vector (IV [or nonce]), a plaintext content, and optional additional authenticated data (AAD).

AES-GCM generates two outputs: a ciphertext and message authentication code (also called an authentication tag).

The nonce [IV] is generated by the party performing the authenticated encryption operation. Within the scope of any authenticated-encryption key, the nonce value MUST be unique. That is, the set of nonce values used with any given key MUST NOT contain any duplicate values. Using the same nonce for two different messages encrypted with the same key destroys the security properties.

AAD is authenticated but not encrypted. Thus, the AAD is not included in the AES-GCM output. It can be used to authenticate plaintext packet headers [for example].

RFC 5084: Chap. 1.5

A typical example of how standards that are primarily important to security libraries also provide valuable information for the application.

The curtain falls.

Epilogue

There is never enough time to implement a perfect product in all aspects.

The Economic Dilemma

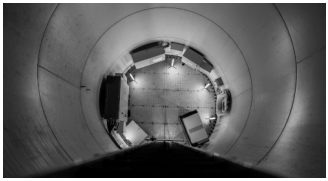
A perfect product also includes complete documentation, which the BlockKeeper open source code technically contains. But the author can't provide it at this point due to time constraints.

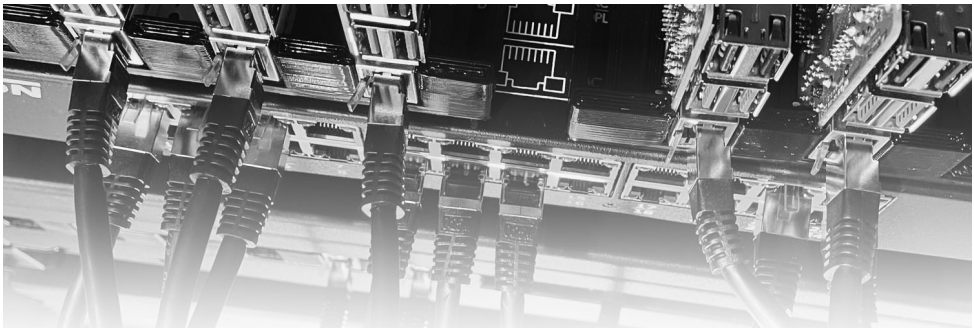
This is how this drama ends.
Maybe not as a tragedy. But also not as a comedy.

Please / Thank you

Support this drama:

- As cofounder of Micro-Colocation.com, the author‘s focus since 2021 has been on green edge computing for small computers. This colocation service was recently released and is happy to welcome any Raspberry Pi, Odroid, Jetson Nano and other Single Board Computer enthusiasts. CC licensed texts like this drama are created within the working hours of this company. You can support the text production by sharing the Micro-Colocation.com link in your network and, of course by booking colocation services.
- Inherent in technical documents are errors of any kind. Although rather forgivable in a drama, comments, corrections and other constructive criticism are always welcome.
- Especially welcome is help with the translation of the German original document into readable English and gladly other languages. In this context, thanks to the free online service deepl.com, which was of great help with the English translation.





Micro-Colocation.com
Your micro computer (with accessories) in a wind turbine or a solar park

Many thanks.

dck

http://dck.one, txt@dck.one

Props

Font

Masks