Countermeasures

- Backups
- Wifi-Sd-Cards
- Data-Hygiene
- Communication-Encryption
- Passwords

Backups

> [!toc] Table of Contents

Much has been written about why backups are so important, and better and worse jokes have been made about the curious fact that everyone knows this but no one seems to do it.

> [!success] Fact is {static} We need backups!

Backups vs. data hygiene

The biggest problem is often that we are not really aware of how much data we accumulate over the years and how important it still is to us. In our article on data hygiene, we advocate the credo of accumulating as little data as possible. But of course, this applies above all to data that is only of temporary use and would otherwise be forgotten in some corner - until it is rediscovered during a police raid.

As important as it is to store as little data as possible, we are all dependent in some way on the state and its institutions:

Official documents, health insurance, bank records, employment contracts, school records, and so on are all things that we may need from time to time to avoid sinking into complete poverty.

Perhaps even more important, however, are photos of our comrades and companions, letters from them, diaries, gifts, and other mementos. All of these are things that could cause immeasurable pain if they were suddenly gone.

Not everything listed above can be easily digitized, but most of it can.

We invite everyone to imagine that all the devices on which their passwords, photos and other personal data is stored are suddenly broken, go missing or are confiscated. Could you deal with that?

Data hygiene & backups go hand in hand

In the section above, we contrasted data hygiene with backups. Here, we would like to argue that both concepts should be considered together so that one benefits from the other.

Devices and accounts that have been used for many years, such as iCloud, Google Drive, WhatsApp, Signal etc., are often overflowing with ancient data that you don't even know still exists.

How much data do you have on your devices that you haven't deleted because you thought, “I might need it again someday”?

The amount of data that you have accumulated over the years is often far too extensive for you to quickly review and clean up.

However, if you had an encrypted backup of all your files, you could clean up your daily-use devices much more easily. Then, you would only carry around what you really need.

> [!success] Success {static}
>
> Practicing data hygiene is enabled by making regular backups!

How To Backup?

We are currently working on a guide to backups for the “Instructions” category, which we will also link to below.

Wifi-Sd-Cards

> [!toc] Table of Contents

Photographers in particular often face the problem that their newly captured images are stored unencrypted on their camera's SD cards until they can transfer them to encrypted hard drives once they are back at their laptops. Added to this is the problem that storage media such as SD cards, USB sticks, and SSDs are very unreliable or even impossible to delete securely if the data on them was unencrypted.

So-called Wi-Fi SD cards can help in solving this issue. Like normal SD cards, they are simply inserted into the camera's SD card slot. However, they do not actually store any images. Instead, they are connected to a mobile phone via Wi-Fi (direct) using an app and send every image taken immediately to the mobile phone. Since your phone is (ideally) encrypted, the data is then better protected.

Data-Hygiene

> [!toc] Table of contents

Whether it's network surveillance, digital forensics, or house searches: surveillance is always about data that could potentially be used against you. That's why it's important to regularly ask yourself what data is really necessary:

- Do we need to take notes for this meeting?
- If everyone was at the meeting, notes of it may not be necessary for some things.
- Do I need to bring my mobile phone with me?
- Do I need to text my friends about the cool thing I just did?
- Bragging has been the downfall of many!

If there is no data, no one can access it. However, the assessment of a few people that certain documents are no longer necessary and that they can be destroyed may be deeply regretted a few years later. Still, depending on the stored data, simply storing documents under the bed or on an unencrypted USB drive might be too risky. So, how can data be stored securely? In any case, only in very few cases on paper!

> [!warning] Warning {static}
>
> If you have created “incriminating material” - Get rid of
>
> it ASAP!

However, most people are probably aware that simply deleting files does not mean that the data is irretrievably lost. Not even when Windows warns you that emptying the recycle bin will really make everything disappear into a black hole forever.

Deleting data securely

> [!tip] TL;DR {static}

> The safest way to delete data, is when the drive is encrypted.
>
> In those cases, every forensic tool still need the encryption password before they can read anything - even if you just deleted the files "normally".

To illustrate what happens when files are deleted “normally,” here is a metaphor:

> [!technical] SSDs vs HDDs
>
> The following scenario only applies to a limited extent to
>
> common types of storage, such as classic HDD hard drives! There are additional things to consider for flash storage such as SD cards, USB sticks, or SSDs. More on this under “Special features” below.

How Files are "deleted" - Anna & Arthur's shared apartment

Anna & Arthur live in a shared apartment. Their names and addresses are listed in the address book (unlike a phone book, everything is sorted by address here). The apartment is the storage medium (hard drive, USB stick, SD card, etc.) and Anna & Arthur are the data on that storage medium. The good old paper phone book (these huge books, where every ones landline number and home address could be looked up at) is the so called *address management system* of the storage medium.

If you want to find Arthur, you enter Arthur's address. The computer then goes to the address, fetches Arthur from his apartment, and displays him on the screen. This is normal operation when data is stored in memory and is being used.

Unfortunately, during the last action, Arthur's mask slipped down over his nose, he was identified, and now he has to leave quickly: The data must be deleted.

If you now click on “delete,” this file will be moved to the recycle bin. Nothing is really deleted when moved to the recycle bin; just think of it as a “files to be deleted” folder.

So you empty the trash can too. What has happened now? Has Arthur disappeared?

No, you have only deleted Arthur's name from the address book. Arthur himself is still sitting on his couch waiting for something to happen: The data is still physically on the storage medium. It is just no longer indexed in the memory's address directory.

If the cops look in the address book, they won't find Arthur's name anymore. But if they simply search street by street, door by door, they will eventually come across Anna & Arthur's shared apartment, where Arthur is still sitting.

The solution? Overwriting the data: Anna & Arthur need random new tenants.

> [!tip] Overwrite data! {static}
>
> In summary: Data is only truly deleted when the addresses in the memory where it was stored have been overwritten by other random data.

However, this process is not standard in any common operating system (whether PC or mobile phone), as these only delete the address entries for the files. This therefore requires additional actions.

Special characteristics

- **Addressing:** With flash memory such as SD cards, USB sticks, or SSDs, the operating system does not know exactly which bits the data is actually stored on. There is no clear connection between physical bits and externally addressable sector addresses. Therefore, these bits cannot simply be overwritten because it is not clear which ones should be overwritten.
- **Overprovisioning**: In addition, these types of memory block certain address spaces from external write access, known as “reserved blocks.” This overprovisioning has three main functions: error correction, optimization of write speed, and preservation of the storage medium's service life.

> [!technical] Technical Details - Overprovisioning
>
> - Error correction: If individual storage cells become defective (e.g., due to wear), the controller can fall back on this reserve to prevent data from being stored “corrupted.”
> - Write speed: Since the reserve blocks are already available “empty,” cells do not always have to be deleted before they can be rewritten. The controller can thus directly access empty cells and write to them immediately.
> - Lifespan: By rotating the data on the memory cells, overprovisioning prevents individual cells from remaining in the same state for a very long time. This typically causes these cells to become asymmetrical in terms of their “on” and “off” states. They therefore tend to tip in one direction or the other. This leads to errors during write operations

> because a transistor that has been "on" for years, for example, is now told to switch to “off” with an extremely short pulse. However, this may not happen because it has been “on” for so long.

Therefore, it is not sufficient to overwrite memory cells with random bits using conventional methods. This leaves the reserve blocks untouched, from which old data can be reconstructed in case of doubt. The ATA specification provides two commands for this: `SECURITY ERASE UNIT` and `ENHANCED SECURITY ERASE UNIT`. The former overwrites with zeros, the latter with random bytes. If these commands are applied to an SSD, the reserve blocks will also be overwritten. Command line tools are available for this purpose in both Linux and Windows, but they can be a little hacky. Most SSD manufacturers such as Samsung, Kingston, Western Digital, and others provide their own tools for this purpose, which can be used.

These tools basically do nothing more than apply these commands to SSDs with their own (proprietary) firmware.

Deleting encrypted data

A more efficient method is encryption. The following applies to both rotating disks (HDDs) and SSDs:

When the data carrier is encrypted, a key is generated and stored in the header of the memory. You will be asked to set a password for the encryption. This password is then used to encrypt the key stored in the header - not the data itself.

Every data read or write operation is symmetrically decrypted or encrypted using the key.

Due to the mathematical properties of modern encryption algorithms, the bit states on the physical data carrier cannot be distinguished from random bits. An encrypted data carrier therefore looks exactly the same as one that has been randomly written to.

To securely delete this data, only the key in the header of the data carrier needs to be deleted and overwritten. This not only saves a lot of time (it takes only a few minutes), but also preserves the life of the data carrier. Completely overwriting a 1TB HDD can easily take more than 5 hours.

More detailed information can be found here.

> [!info] Summary {static}

- Data on unencrypted data carriers: deleted data leaves traces that can be recovered. Therefore, data must be overwritten with random bits (preferably several times) when deleted.
- Data on encrypted storage devices: These can only be decrypted using the key in their header. This key is secured with a password. If only this key is deleted and overwritten, the data can no longer be recovered.

Communication-Encryption

> [!toc] Table of Contents

The encryption of all communication plays an essential role in our digital lives. In this article, we want to explain what communication encryption means, what different types of encryption exist, and what advantages and disadvantages they have.

We distinguish between transport encryption and end-to-end encryption (E2EE).

> [!info] TL;DR {static}
>
> While transport encryption is a nice-to-have, it is in no way sufficient for most use cases - we recommend using end-to-end encryption (E2EE) whenever possible.

Transport encryption

Transport encryption is generally implemented with SSL/TLS. Those are encryption-based Internet security protocols that provide privacy, authentication, and integrity to Internet communications You are using SSL/TLS everyday in your browser, for example, when a padlock appears next to the URL and `https` appears before the URI. If this is not used, only `http` appears (and in most cases a warning appears that the connection is not secure).

We will use the graphic below and a practical example to explain how transport encryption works.

Example: Mail with transport encryption

Anna wants to send Arthur a message, for example by email. The example also works with other services without E2EE, such as Telegram, Discord, or chats in games. However, then there would only be one server instead of two.

Here is the example with email:

Anna has an email address on the yellow server; in our example, it would be `systemli.org`. Her email address is therefore `anna@systemli.org`

Arthur has an email address on the red server, in this case `riseup.net`. His email address is therefore `arthur@riseup.net`

Because we are talking about transport encryption, neither of them uses E2EE such as PGP. This means that Anna does not have Arthur's PGP key, and vice versa!

The keys and locks in the graphic below symbolize so-called **certificates**. Each server has its own certificate with which communication to it can be encrypted. Only the server in possession of the certificate also has the corresponding key an can read the information that is sent to it.

If Anna now wants to write an email, she retrieves the certificate from Systemli (yellow lock) and uses it to encrypt her email. This is completely independent of who the email will ultimately be sent to! Arthur's receiving address (mailto:arthur@riseup.net) is then written on the "envelope", just like with normal mail. This email (yellow, sealed envelope with a lock) is then sent to the Systemli mail server (yellow box).

The Systemli mail server now opens the email encrypted with its own certificate and scans it for spam, for example. Above all, it looks at the recipient address on the envelope: `arthur@riseup.net`. The server recognizes the part after the `@` symbol as the mail server to which it must forward this email: `riseup.net` (red server). So it quickly goes over to Riseup, grabs a copy of their certificate, encrypts Anna's email again with it, and sends it (red, locked envelope with lock) to the Riseup mail server.

From here, this process repeats itself until the email reaches Arthur. The Riseup server unpacks the email, repacks it, and finally sends it to Arthur.

Transport encryption graphic

Problem

The problem here is obvious. Every participant in the communication chain can easily open and read the email. In addition, many applications (as listed above) store copies of the messages on their (email) servers. See more about network surveillance

End-to-end encryption

Once you understand the threat posed by transport encryption, the need for end-to-end encryption is almost self-explanatory.

1. Anna obtains Arthur's lock (public key). This point is very important; please note the [TOFU] section!
2. Same as in 1.
3. Anna encrypts her message with Arthur's public key.
4. The message remains encrypted in all steps of 4 (a-e). Only the metadata (e.g., sender/recipient address) is visible (at all possible points, including during transport!) and is read by the servers in order to forward the email.
5. Arthur receives his message. Because the message was encrypted with his padlock and he has taken good care of his key (private key), only he can decrypt the message with his key.

End-to-end encryption graphic

TOFU is bad

TOFU: Trust On First Use

The key must be verified “out of band.” An unencrypted (i.e., transport-encrypted) email makes the exchange of public keys vulnerable to interception. This is called a "machine-in-the-middle attack".

Graphic machine-in-the-middle attack

For more information on the dangers of transport encryption, see network surveillance.

Passwords

> [!toc] Table of Contents

Good passwords are one of the most important countermeasures against data leakage. We will discuss what constitutes a good password below.

- passwords secure encrypted data, such as: hard drives and password databases
- passwords secure access to online accounts: against non-authorities!
  - authorities like the police, might be able to get a court order, because your data is stored there unencrypted, so they don't need your password.

> [!success] General rules {static}
>
> - **Do not** reuse passwords
> - Use **strong** passwords
> - Use **two-factor authentication**

That's why it's our duty as modern activists to use a password manager. It helps us meet all these requirements without much difficulty. This way, we not only protect our own access, but also the information behind it that is linked to our comrades!

In this article you will find explanations on:

- What a password manager is capable of
- What a good password is
  - What second factor authentication means and why it is recommended

Password managers

A password manager stores all your passwords in a single encrypted database *(which is just a file)* protected by a **master password**. This means that your passwords are not stored in plain text on your system or on paper in your home, and you don't have to remember them all yourself.

Since you no longer have to remember passwords yourself, it is not a problem and is also recommended that you generate a separate, strong password for each account. This is very easy to do with the password manager itself.

The password manager also stores the assignment to websites & apps for which you have generated the respective password. This also makes phishing more difficult, because the password will not be displayed as a suggestion on a false URL.

As mentioned above, the password manager itself is protected by a strong master password and/or other factors (see below 2-factor authentication). This is therefore *(apart from hard drive encryption)* the only password you really need to remember and can therefore be a little more complex. The rule is: it is better to remember **one strong** password than **many insecure** passwords.

> [!tip] Which password manager? {static}
>
> Read more about this in the recommendations for password managers.

Strong passwords

Okay, but you still need at least one strong password for the password manager. But when is a password strong?

An important basic requirement is that the password is generated randomly. Anything you come up with, no matter how clever your system may be, should be considered insecure.

Optimized algorithms enable authorities to search specifically for possible passwords used by activists by trying out vocabulary, quotes from revolutionary writings and songs, etc., while saving time and energy by avoiding fascist vocabulary, for example.

“History is a history of class struggles” may have seven words, but for the reasons mentioned above, it is a very poor password!

Here you can already see that the term **passwords** also refers to **passphrases**. Passphrases are randomly generated strings of words. They have the advantage that people can remember much longer strings of characters.

> [!tip] A good password is {static}

> - A good password should have an **entropy of more than 120 bits**. This is a bit more than the usual recommendation of 80-100 bits, but therefore should be safe against future hardware advancements.
> - In practice, when speaking about passphrases, this should end up at at least 5 words long, **preferably 8.**
> - Created using a password manager or Diceware *(dice and a list)*.

We explain below where the numbers 5 and 8 come from. There, we look at how long it would theoretically take to crack a randomly generated password or passphrase. However, these tables are always subject to many ifs and buts.

2-factor authentication

2FA ensures that simply entering a password is not sufficient for complete authorization, as it is assumed that passwords may be corrupted. Therefore, a second instance is requested for complete authorization.

The recommendations for password managers include an example scenario showing how a KeePassXC database can be secured with a second factor.

The second factor can be based on various characteristics:

2nd factor: Possession

You need a special device that either displays a number or must be plugged into the computer via USB. If the attacker does not have this “device,” authorization will not be granted. (Hardware tokens, *2FA apps*, *SMS*)

TOTP software

TOTP stands for *time-based one-time password* and can be set up with all common password managers such as KeePassXC. The login process then consists of entering your username and password, after which you will be asked for the TOTP (e.g., a 6-digit PIN), which changes every ~30 seconds.

USB hardware tokens

These look like normal USB sticks. If a service/hard drive or similar configured with this token is to be unlocked, the stick must also be inserted into the device being used. These tokens are often protected with a PIN, so stealing them is not enough. The number of PIN attempts is limited.

Since all of this is implemented and protected at the hardware level, it is a very secure method of authentication. *(The relevant standard for security tokens of this type is called FIDO2, the old standard is U2F.)*

TOTP hardware tokens

Similar to `TOTP software`, but not in an app such as a password manager, but as a thumb-sized device. They have a small screen that displays the 4-6 digit `TOTP`, which changes every ~30 seconds.

When logging in, the code displayed at that moment must always be entered as 2FA. However, the standards for tokens of this type are usually not open source, which is why we do not recommend using them.

SMS

Probably the better-known method. To verify the identity of the user, the respective service sends an SMS to the phone number registered with the account. Since the mobile network cannot be considered secure, we do not recommend this method.

2-factor biometrics

Unique biometric characteristics must be verified during registration (fingerprint, facial recognition, iris scan). Biometric authentication is particularly widespread for smartphones.

However, biometric authentication poses a problem for us in that the authorities can simply use our biometric characteristics under duress and by force. We therefore advise against biometric unlocking methods such as fingerprint and Face ID as a matter of principle.

> [!technical] Technical
>
> Bio-metrics such as fingerprints or facial recognition have been proven to be falsifiable. Starbug from the CCC has already demonstrated how easy this is for fingerprints, faces, iris and vein recognition. The most important point here, however, is that you can never change your bio-metric characteristics. A corrupted password can be reset. A fingerprint or face, however, cannot.

The only exception to this is GrapheneOS, which offers a PIN as a second factor limited to twenty attempts for fingerprint recognition and otherwise meets the highest security standards.

2nd factor: Knowledge

For example, the security questions that were common in the past, such as “What is your place of birth?” However, these “security questions” usually imply answers that someone who knows you well could easily find out for themselves. We therefore don't recommend them.

Time to crack

In reality, how long it takes to crack a password depends very much on the exact circumstances. The calculations here assume a very specific scenario. The scenario shown here assumes very favorable conditions for the attackers. This means that in practice, it will take even longer.

Time to crack a password

It should also be noted that these times are for *one* password from *one* person. All of the hardware is busy with this task, so no other passwords can be cracked during this time.

> [!technical] Technical
>
> We assume an MD5-hashed password and that the attackers have access to the hardware used to train ChatGPT: 10,000 NVIDIA A100 GPUs. Purchase price: approx. $9,000 per unit *(2024)* for the cheaper version with 40GB of memory. That's a total of $90 million. Even renting this amount of hardware is not cheaper in the long run. Further details on the scenario can be found at hive-systems, who performed the calculations.

Important prerequisite: The password must have been generated randomly! This means that this is purely *character brute forcing*. So, for example, you start with `0000` and try:

- `0001`,
- `0002`,
- `...`,
- `AAAA`,
- `AAAB`,
- `...`,
- `A-A-A-B-B`,
- `R€70lut10n`,
- …

etc. **without** word lists optimized for the target person.

a table shows the amount of time to password-cracking, according to above described scenario

Time to crack a passphrase

However, a random password that is sufficiently long and contains letters, numbers, and special characters is difficult for humans to remember. That's why we recommend using **passphrases** for passwords that you need to remember, such as those for your password manager and the hard drive of your computer and phone. These consist of words instead of individual letters. They are much easier for humans to remember, but are no less secure than passwords. See also: xkcd 936

> [!technical] Technical
>
> In information theory, it must always be assumed that the attacker knows how we created the password in order to evaluate its security. Therefore, the attacker uses a word list attack here. Otherwise, everything remains the same.

For example, the word case is assumed below, namely that the attackers know exactly how many words from which language and in which format *(i.e., upper/lower case, which characters between words)* were used for the password and that they use *(in the left column)* the hardware used to train ChatGPT to crack it. Therefore, these graphics should not be taken at face value.

a table shows the amount of time to passphrase-cracking, according to above described scenario

As already mentioned, random passphrases can be created using password managers or, in a similar way, with dice and a word list that is as large as possible.