ESPERANZA

No. 1

THE NON-PERIODICAL OF THE REVOLUTIONARY TECHNICAL COMMITTEE - IN THIS ISSUE: DIGITAL PRIMER FOR COMMUNIST ORGANIZATIONS. ISSUE 1, DECEMBER 2023.

PREFACE – OUR INTENTION

We open the first issue of this non-periodical with a varied guide on topics related to the digital realm that we believe may be important to organizations. While this mainly focuses on the basic to lower-intermediate level of digital security, we also make a few remarks on content creation and hosting in the latter part, mainly aimed at publications.

Our belief is that digital literacy is one of the most important aspects of organizing in the 21st century. While many communist organizations understand that they must use new information and new technologies to inform their political practice, most of them seem to not seek a deeper understanding of modern software beyond “pop science” conceptions.

Let’s use a practical example. Everyone knows that they are under constant, passive surveillance. If not by the government, by companies which seek to map their habits into a profile so that they can deliver you ads more efficiently. But what do most people know beyond that notion? Nothing.

The bourgeoisie feels free to announce to the world about how ubiquitous its surveillance is: they tell everyone who will listen about their profiling algorithms, about facial recognition, about how everything is backdoored. They won’t tell you about the ways to mitigate the effectiveness of their widespread measures, however, and that is where the crux of their political power resides. The only mitigations that are proudly advertised are commercialized ones: VPN services, “private” e-mail, etc. Which do offer some level of protection, but are functionally useless against actual surveillance without prior knowledge of their limitations.

Needless to say, technology is not magic. It is the duty of communists to smash mystifications, and today, we make a small effort at demystifying the reach of surveillance, by bringing together some basic and intermediate notions of digital security, along with notes on the accessibility of communist writings.

BASIC COMMUNICATIONS AND DATA SECURITY

Encrypted messaging

Let's talk about encryption. In recent years, it has become a buzzword, in no small part due to the mysticism we condemned in the preface. We shall explain what it is, and what problems it solves.

What are digital communications, after all, if not just a stream of bytes? Their nature makes it trivial for an actor to inspect information if they have access to any of the infrastructure it moves through. Security researchers consider this an opening for a "man-in-the-middle" attack. Inspecting digital communications is easier than listening in to phone calls, as information can merely be hoarded, with algorithms assimilating the data and sweeping it to find any activity an actor desires.

Digital communications can be inspected by a variety of actors, not just the State or an Internet Service Provider. A router at a café or library, for example, may be configured to create a copy of everything that goes through it. A popular way to target protesters is setting up a receiver that poses as a cell tower, and reroutes data to a real tower while letting an attacker collect what's going through the device. Methods for snooping around are easily accessible to any moneyed and dedicated bad actor.

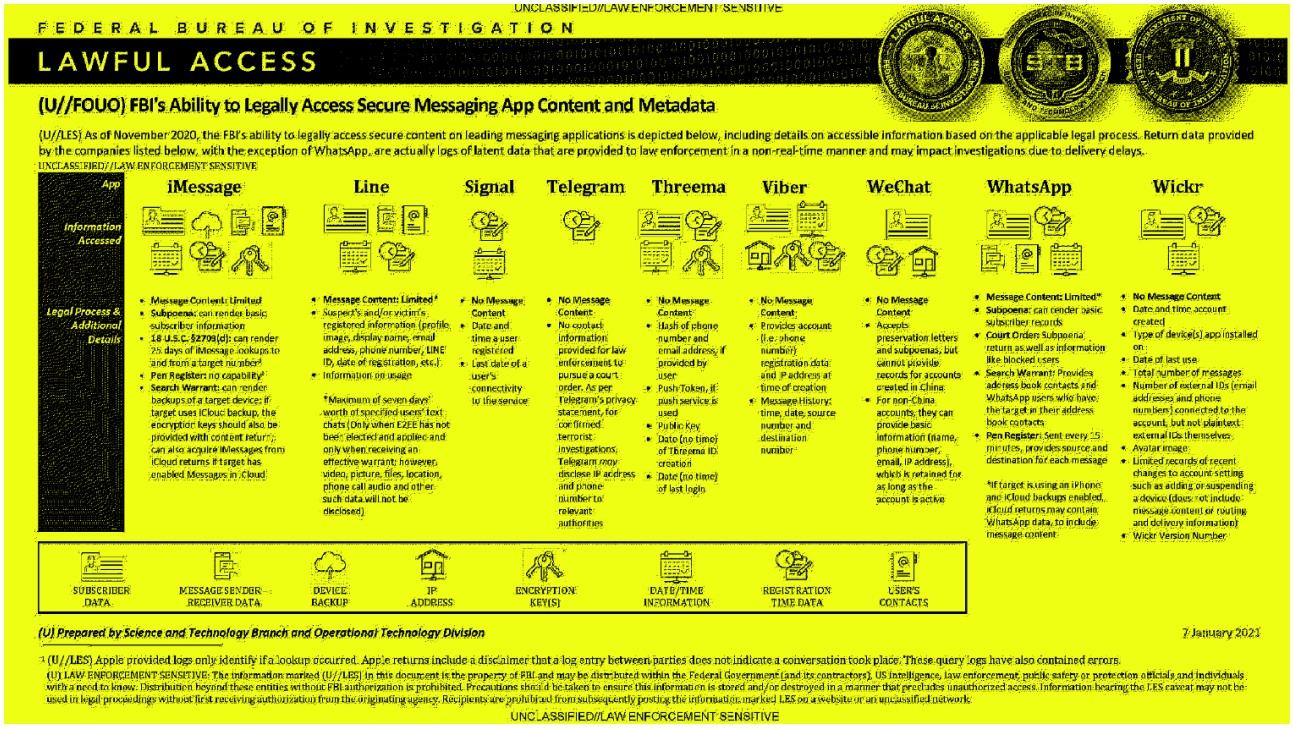
What encryption does is make the contents of the data unreadable while in transit, making it so that only both ends of the communication can know the contents. A bad actor will know the data is encrypted, but won't be able to do anything about it.

Encryption nowadays is widespread, with most internet traffic being encrypted. The main issue when using online platform X or Y for communicating is not related to the data being exposed in transit from you to the service or from the service to you, it is instead related to how much data the platform has on-site about your messages, with the worst-case scenario being that they're just storing plain-text copies of them.

So, our main concern in secure communications is not with encryption in itself, but "end-to-end" encryption. Essentially, we want the contents of a message to not be accessible to any intermediates, not just in the path to the first intermediate.

There is one king of end-to-end encryption when it comes to messaging, and that is Signal. It has gone through the trials and tribulations of legal requests before, and this slide obtained from a Freedom of Information Act

request reveals just much the FBI can get from a subpoena directed at it vs. other messaging apps:



Pretty straightforward. If an organization needs an easy-to-use app, that can be installed by all of its members, Signal is the best option. We know of a few organizations which have already moved to it, and it has been a success, with even older members who didn't really have an opportunity to learn digital literacy being able to use it easily.

Sensible data storage

Every organization eventually handles the data of its members. This can lead to a varied amount of issues. The issues we fight ourselves battling today in regards to data storage exist ever since paper records became a thing. If an entire folder is stolen, what information will someone have when they read through it? We can't have an entire organization be compromised due to a flash drive being forgotten at a restaurant.

Thankfully, encryption is accessible in the digital realm, and we will have a walkthrough on encrypting files in the next section. However, we will pre-empt some reckless data storage practices now.

We have prepared a spreadsheet with the most common kinds of personal information and our recommendations for storage: what you can store, what should only be an interpersonal or organizational matter, and how

that information should be transferred. As every organization is different, this should be not seen as a rigid guideline.

Information categorization spreadsheet

Type of information	Can be stored (org records)	Can be stored (In individual members' devices)	Can be transferred + method	Reasoning
Phone Number	No	Yes	Yes, messaging apps	People save and share each other's contacts, but no official records should be maintained.
First Name or Mononym	Yes	Yes	Yes, via any means	Insensitive. A pseudonym is a good idea, however.
Middle and Last Name(s)	Avoid, only for financial purposes	No	Only PGP encrypted, with rigid storage and erasure practices	Treat these records with extreme care.
E-mail	Yes	Yes	Yes. Between members, preferably via secure messaging	Insensitive. Vector for fraud attack.
Employer & Occupation	Depends, read reasoning	Depends, read reasoning	Depends, read reasoning	Obviously useful for labor organizing. Sharing it requires critical thinking.
Birthday	No	Avoid storing side-by-side with name	Yes, all messaging	Members may celebrate it, but no records should be kept
Address	No	Read Reasoning	Only PGP encrypted, with rigid storage and erasure practices. Sometimes via secure messaging, read reasoning.	Only share when location being used as meeting place.
CC Number, Expiry, and 3-digit code	Avoid, only for financial purposes	No	Only PGP encrypted, with rigid storage and erasure practices.	Treat these records with extreme care

USER-MANAGED ENCRYPTION: HOW TO USE PGP

When Signal isn't enough

In a situation where we are dealing with communications in the political center of an organization, and especially where we are dealing with critical information, merely using a secure messaging app isn't enough. Instead, stronger encryption, where the user controls their own encryption keys, must be used.

The common user having access to such a level of key management and security can only be achieved today with PGP. What follows is a basic walkthrough on what PGP is, and how to use it.

What is PGP?

PGP stands for Pretty Good Privacy, it is the name of a program originally created in 1991, but nowadays, the acronym is used to refer to any software that conforms to the OpenPGP specification. PGP is a way to encrypt and authenticate messages and files, mostly for transferring data over a network where you know someone will have access to the contents of a message.

OpenPGP is the best anyone outside of a big corporation or the State will get out of an encryption management tool, and the best algorithms it supports are currently unbreakable for standard supercomputers. Even a full-fledged quantum computer may still take months to break them.

How does PGP work, if, for example, a user wants to send an encrypted e-mail to another user?

Envision two people: Leon and Natalia. Leon wants to send a message full of specific compliments to Natalia, and they both want to make sure nobody can see it. In that case, of course, Leon would want to encrypt the message before sending it. There are five basic steps to actually encrypting and sending a message using PGP:

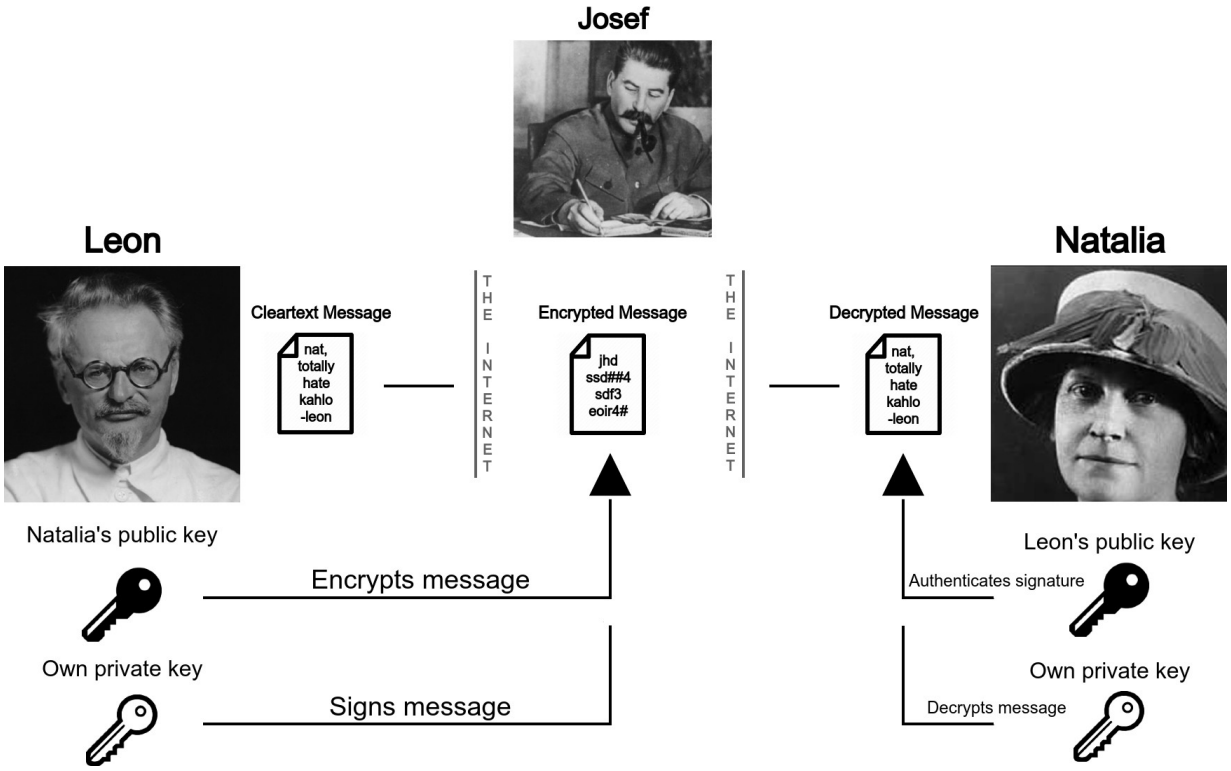
1. Both Natalia and Leon would have to create a public and private key. Together, these keys are called a "key pair".
2. Natalia would share her public key with Leon, ensuring she keeps her private key safe.
3. Leon would take the public key he got from Natalia and encrypt the message he wants to send over to her.
4. Leon would cryptographically sign the message with his own private key, proving its authorship, and send over his public key to Natalia.

5. Leon would send the message over to Natalia, and she would decrypt it with her private key.

Let us suppose we have a bad actor, Josef, seeking to view the contents of the message. During transit, Josef would be able to know Leon's message is encrypted, however, he would not be able to do anything about it. Josef would have to trick or physically extort a party to be able to have access to their private key, and, if Leon decided to send an unilateral message that only Natalia can decrypt, he would have to target Natalia.

It's also worth noting that Leon can double down on this unilaterality, and also leave the message unsigned, completely taking his private key out of the equation. This could serve to deny responsibility for the message's contents if Leon is proven to have a previous pattern of signing every one of his messages, but it also removes any cryptographic guarantee to Natalia that the message is actually from Leon, so she would only be left with a (falsifiable) mailing address as proof.

Below is a basic illustration of the dynamics described:



GnuPG and Kleopatra

The specific implementation of PGP that this guide will be using, and the most widespread and trusted one, is GnuPG, which stands for "GNU Privacy Guard". It implements industry-standard and strong encryption algorithms, and its development is overseen by many specialists.

GnuPG only provides a text-based interface, so, for ease-of-use, we will be using Kleopatra, a graphical interface which exposes GnuPG's functionality in a user-friendly manner.

Windows Installation

Note: Windows is not recommended, for obvious reasons. While, by all accounts, Windows doesn't send any more telemetry data than you tell it to, you should not count on this behavior being true forever. There is also no telling if Microsoft can start gathering more data on you if the government specifically requests them to target you.

Download the official gpg4win package from https://www.gpg4win.org/. Windows' User Account Control should automatically authenticate the code signature for you. If it marks the software as coming from an unknown source, this means the binary you downloaded is unsigned – check the gpg4win news section to see if that's something you should worry about.

Once you have started the installation, you will be met with the standard install wizards you see in many Windows applications – there's no secret here, just click "Next" until the installation is completed.

GNU/Linux Installation

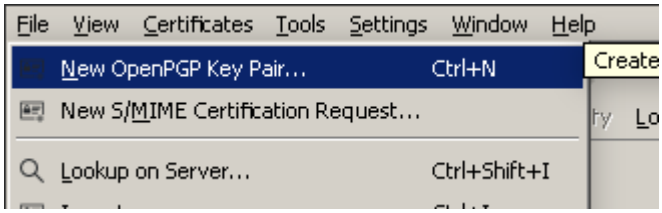
Install Kleopatra via your favorite package manager. Installing Kleopatra will automatically install GnuPG, as it is a requirement. On Ubuntu, Mint or any Debian-based distribution, it should be under the name `kleopatra`. It's also under that name on Fedora/RHEL repositories and on the Arch repositories. For Arch distributions that do not use the standard repositories, there is a slightly outdated version on the Arch User Repository called `kleopatra-git`.

Distribution	Installation Command (as root)
Debian	`apt install kleopatra`
Fedora/RHEL	`dnf install kleopatra`
Arch	`pacman -S kleopatra`

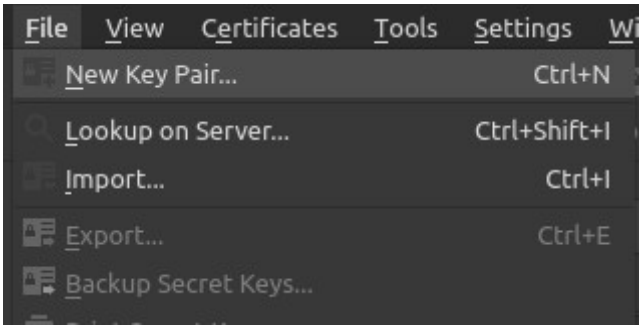
Creating a New Key Pair

Once you open the software, creating your public and private key is very simple: In the Linux version, click the “File” menu, click “New Key Pair”, and select “Create a personal OpenPGP key pair”. In the Windows version, click the “File” menu, and select “New OpenPGP key pair…”.

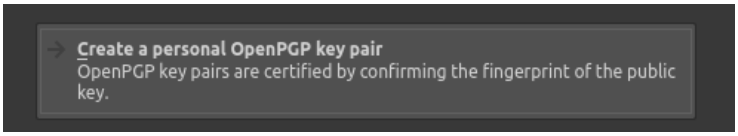
Process on Windows – Single Step:



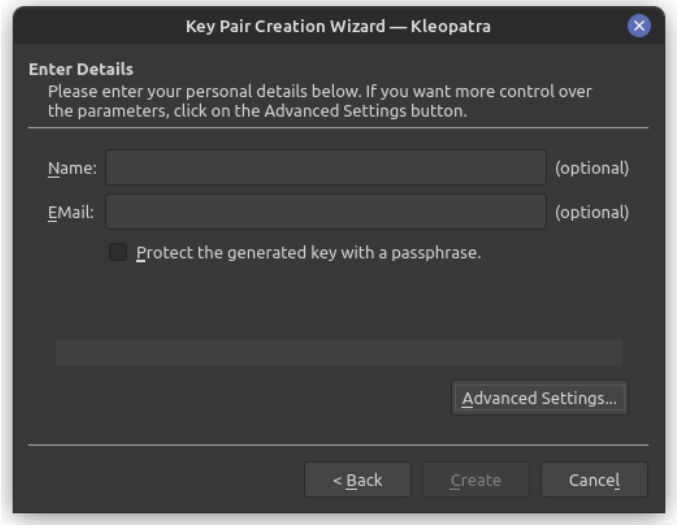
Process on Linux – Step 1:



Process on Linux – Step 2:

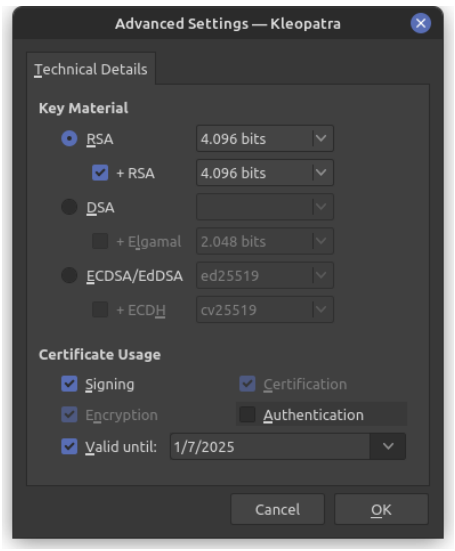


After following those steps, you will see a menu like this:



Put your desired name and e-mail address. **Remember, PGP is pseudo-anonymous, encrypted data for which you are the receiver will have your key metadata attached to it**. So, choose a pseudonym, and maybe leave the e-mail field empty, unless you benefit from such metadata being verifiable. You may also choose to protect your key with a passphrase – that's up to you, and is highly recommended.

Next, click the "Advanced Settings" menu, and ensure you have selected the RSA + RSA encryption type, both at 4096 bits. You may also select an expiry date for your key, or no expiry date at all – since you won't be around forever, it's recommended your key isn't around forever, either. On average, select an expiry date 5-7 years from the current date for your personal key. Once a private key expires, it may be renewed, and new versions of the public key may be distributed. Overall, you should have something that looks like this:



Click “OK”, and before clicking “Create”, think up a passphrase. Do not store the passphrase on your computer!

Be aware that once you create a private key, you should protect the device it was created in. Merely deleting and moving it to external media isn’t enough, as modern file systems store multiple backups of certain files (we will touch on this later). However, it is highly recommended you make a backup of your secret key on inexpensive and easily-destroyable media, such as a CD or even paper. This will be presented as an option after key creation in Kleopatra on Linux. **Do not, under any circumstances, make online backups**, also, **you should store backups DIRECTLY into removable media**. You should only print your private key if you have direct access to the printer and can disable its Wi-Fi functionality.

To export your private key, go to the “Certificates” section (the one Kleopatra is open on by default), select your key, right-click it, and select “Backup Secret Keys…”. A save menu will appear. Save the key on your preferred external media, and store it wherever your paranoia tells you is appropriate.

Importing a Public Key

For many operations involving message and file decryption/encryption, you will need the recipient/sender’s public key. Public keys normally have a .asc extension, and on Windows, Kleopatra sets up a file association for them, which means that you can import them by double-clicking them.

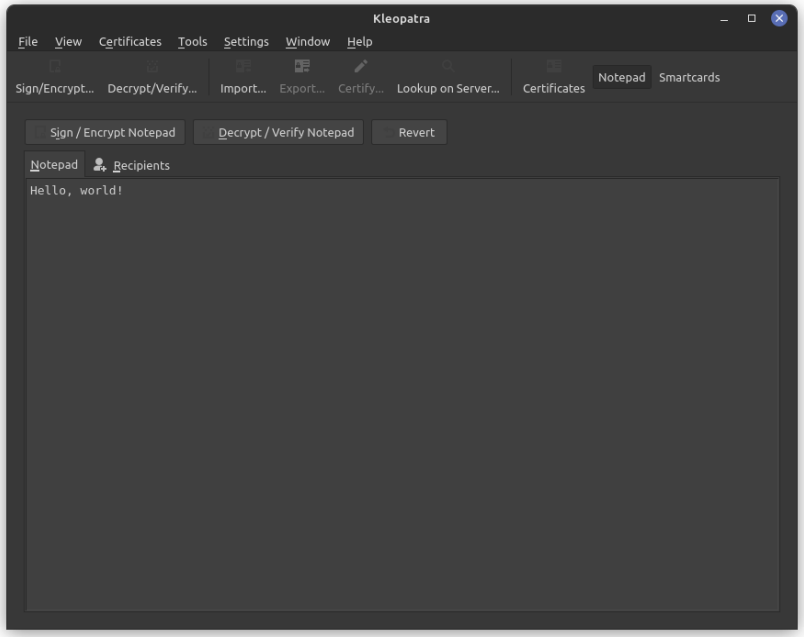
To start importing a public key, click the “Import…” button on the top of the window, and select the desired key file. The key will automatically be imported, and you will be asked to certify it. After that is done, in the default “Certificates” screen, select the key, then right-click it and select “Change Certification Trust…” (on Linux) or “Change Certification Power” (on Windows).

You will be faced with a range of trust options, it’s up to you to select which one you think is appropriate. The level of trust is important, as PGP relies on trust chains between keys. Read the descriptions for each trust level carefully.

After that is done, you’re ready to send and receive data to/from the owner of the key.

Creating an encrypted message

Create an encrypted message by going to the “Notepad” section, and typing in your message.



After writing our desired message, move to the “recipients” section to configure who you will allow to decrypt the message, and if you will sign it, keeping in mind all previous remarks about anonymity. You may include as many recipients for the message as you want, by using the “Please enter a name or e-mail address…” field as a search box.

Then, move back to the “Notepad” section and press the “Sign/Encrypt Notepad” button. You will be presented with the garbled, encrypted version of your text. Copy the full block of text, and send it via your preferred means.

Decrypting an encrypted message

When you receive an encrypted message, you may decrypt it by pasting it into the Notepad, with the begin and end blocks included, and clicking the “Decrypt/Verify Notepad” button. If the decryption is successful, you will see a success message, the signature of the sender (if it exists), and the list of recipients.

Encrypting and/or signing a file

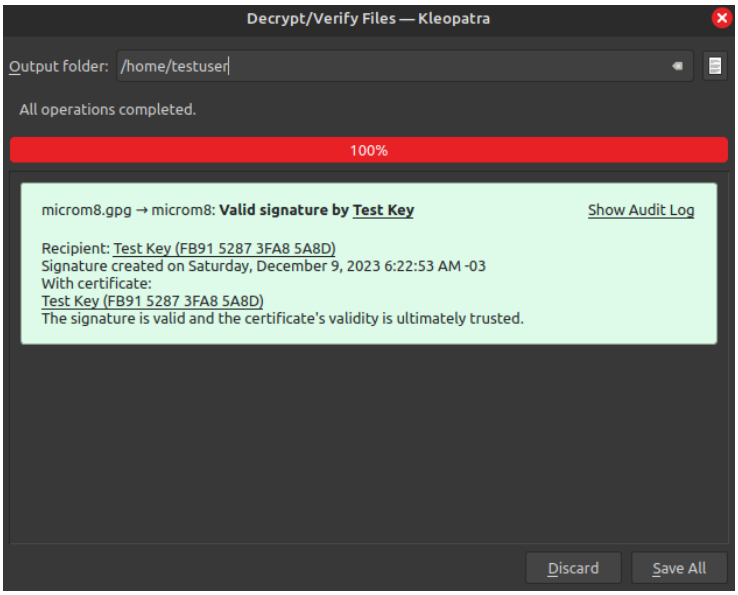
To create an encrypted file, click the “Sign/Encrypt” button at the top left. You will be presented with an options menu. Here, you will be able to decide if you want to merely sign the file or if you also want to encrypt it, along with selecting the recipients. You will also be able to select the path to the encrypted file.

You may also opt for foregoing the key system entirely, and instead just encrypting a file so that a password is required to read it. Wether this is

appropriate depends on the kind of data being handled, so it is up to you to make that decision.

Decrypting and/or verifying a file

To decrypt and/or verify a file's signature, click the "Decrypt/Verify" button right besides the "Sign/Encrypt" button at the top left. You will be asked to select the file you want to operate on. After you proceed, the software will start the decryption/verification process. If it is successful, Kleopatra will tell you. If the file was also encrypted, you will be able to save a copy of it by selecting a path using the top bar, and clicking the "Save All" button.



Sharing your public key

You can export your public key by pressing the "Export" button on the top bar and selecting the path you wish to save the key to. After that, you may share it via any means you find appropriate.

If you want the wider public to be able to send you encrypted messages, the most popular option is to upload your key to a keyserver, the most popular and trusted ones are:

- PGP Global Directory (https://keyserver.pgp.com/)
- keys.opengpg.org (https://keyserver.pgp.com/)
- Ubuntu PGP Keyserver (https://keyserver.ubuntu.com/)

Keep in mind you should NEVER access keyservers trough HTTP, always use HTTPS, as it encrypts your web requests, otherwise, your ISP can alter the content you see and download.

The safest option if you don't want to be too public with your public key is to upload it to a personal website, or upload it to a standard downloads site and link it on your social media profiles.

Revoking a public key

If your private key has been compromised and you wish to inform people of that, or you've stopped using your current key pair, you may upload what's called a "revocation certificate" to the places where you previously uploaded a public key, which is basically a version of your public key that informs the user that it's been revoked, so that when people look up and utilize the most recent version of your public key on a keyserver, it won't work.

You may generate a revocation certificate by right-clicking your key in the certificates menu, clicking the "Details" option and clicking "Generate revocation certificate".

SECURE FILE STORAGE

When dealing with sensitive documents, we must devise a way to safely store and erase them. When it comes to storage, we must prevent a bad actor from having access to the contents of the storage, either over a network, or by physically seizing the storage medium. When it comes to erasure, we must be certain that the data cannot be recovered by normal means, such as data recovery software, and by specialized means such as forensic chemistry. In this section, we will address storage.

While there is specialized hardware for storing large amounts of data, we will assume that the reader of this document does not have the resources to acquire and run such hardware, instead, we will assume that sensitive information will instead be stored in an external medium, such as a secondary SSD, USB drive, SD card, etc. As such, this part of the guide will focus on setting up secure external storage from a machine with a Unix-like operating system (such as GNU/Linux) installed.

Considerations

Although we will only focus on data erasure in the next section, we must consider the proliferation of the data at the time of storage, which means we have to account for how many copies of the same file are produced and how many storage devices it goes through before finally arriving at the target

medium. The goal is to minimize proliferation, as the more we have of it, the harder it gets to truly erase records.

First, the number of copies: modern filesystems (a filesystem is the software that exposes the concept of “files” to your computer) normally keep a log of each individual write operation on a disk in order to prevent data loss in case of a power disruption. This is called “journaling”, and you should assume your computer’s main storage device uses a journaling filesystem. We want to be unaffected by journaling, and so, we will avoid touching the main storage device of the computer at all.

Second, the devices it goes through: normally, when someone downloads a file and copies it to external storage, the file touches at least three devices: it is copied to RAM, then to your main storage device, and then to your external storage device. We want to cut the middle-man, and we want to create/download a file directly to RAM, and have it reach our external storage device without it touching our main storage device. To achieve this, we will want to create a “ramdisk”, which is essentially an area of memory mapped to appear as a folder on your computer.

RAM is non-persistent storage, and its contents are erased just a few minutes after poweroff. This is why we want to move contents directly from RAM to the target device. However, not all RAM is “real” RAM. In order to avoid running out of memory, operating systems move contents in memory that are not constantly being used to a persistent storage device. In unix-like operating systems, this is called “swap space”. If we want to reduce proliferation, we will want to ensure that we are not accidentally using swap space.

Creating a ramdisk

The simple way to create a ramdisk in GNU/Linux is via the usage of a filesystem called “ramfs”, which doesn’t utilize swap space at all.

We will do this entire process via the terminal. First, we must set up the directory in which we will map the filesystem to. In this example, we’ll use root privileges, safely invoked via the escalation command `sudo`, to create the directory `/mnt/ram-staging`:

```
sudo mkdir /mnt/ram-staging
```

If you want that directory to be accessible to you, as a normal user, at all times, you should change its ownership. Replace “username” in the below command with your own system username:

```
sudo chown username:username /mnt/ram-staging
```

Now, let's move onto configuring the filesystem to be mounted at start-up. We can do this via the filesystem tab. Open `/etc/fstab` with root privileges using the text editor of your choice. We recommend using `nano`, an easy-to-use editor that comes pre-packaged in many GNU/Linux distributions:

```
sudo nano /etc/fstab
```

Now, create a new line at the end and add the following, separated either by a tab or by a whitespace:

```
none    /mnt/ram-staging    ramfs    defaults,size=512M    0    0
```

If in nano, save the file with CTRL + S and exit the program with CTRL + X. This will create a filesystem of the type `ramfs` with the device label none, mount it in the folder `/mnt/ram-staging`, using default settings with a size of 512 megabytes, disabling filesystem dumps and boot-time checking.

Restart the machine, and congratulations! Now you have a RAM-based filesystem which doesn't utilize swap space. To check if everything you did worked, change directory to the ramdisk using the command `cd /mnt/ram-staging` and run the following command:

```
df .
```

This will display one line, separated in multiple columns, containing information about the storage device of the current folder. You are interested in the first column. If it says "none", everything is working fine.

Using the ramdisk

Many graphical file browsers shipped with GNU/Linux distributions are likely to show your ramdisk as a separate storage device that you can just drop files into. If they don't, you can just manually navigate to the directory. You can also symbolically link the directory to your desktop using the following command:

```
ln -s /mnt/ram-staging ~/Desktop/ramdisk
```

This will create a symbolic link to `/mnt/ram-staging` named "ramdisk" in the current user's desktop.

It would also be wise to change your browser's configuration to redirect downloads to the ramdisk. As browsers and other file download programs treat downloads pretty much independently, there is no telling how much data propagation is inherent in them, unless you can audit the program yourself.

Keep in mind that the size of a ramdisk is merely a suggestion. The ramfs filesystem will not stop transfers that exceed the stipulated size of the disk. This means that there is the possibility that a large transfer will consume the entirety of your RAM. In such a case, your computer will lock up, and you will lose the contents of the disk upon restart.

SECURE DATA ERASURE

A common computing trivia fact is that "deletion" and "erasure" are two different operations. "Deletion" merely signals to a computer that the storage is free and can be overwritten at any time, while "erasure" is an operation done before the deletion of the file, which aims to overwrite the contents as to make them unrecoverable. Due to its similarity to the physical process, "erasure" is often called "shredding".

With shredding, not only do we take into account the previous considerations about data propagation, but we must also take into account certain truths about the storage device the data is on.

As to not save on facts, this section will be highly technical, and contain an overview of the most famous shredding method and how it translates to the realities of today, possible alternatives, along with the possible weaknesses that can be induced by an attacker.

Gutmann Method

The most popular shredding method pertains to magnetic drives, and was devised by Peter Gutmann in the paper "Secure Deletion of Data from Magnetic and Solid-State Memory", published in 1996. Gutmann was a pioneer in theorizing that highly advanced organizations, such as the government, might have specialized equipment that would allow them to recompose a file erased from a disk by leveraging the physical properties of the storage medium. As such, his paper delves into the physics of hard-disks and devises a pattern for overwriting files based those observations.

Gutmann lays the specific problem pertaining to hard-disks in simple terms:

"In conventional terms, when a one is written to disk the media records a one, and when a zero is written the media records a zero. However the actual

effect is closer to obtaining a 0.95 when a zero is overwritten with a one, and a 1.05 when a one is overwritten with a one. Normal disk circuitry is set up so that both these values are read as ones, but using specialised circuitry it is possible to work out what previous "layers" contained."

As such, a pattern is required to minimize the amount of data that can be recovered via this "specialized circuitry". The objective is, in Gutmann's words, "to flip each magnetic domain on the disk back and forth as much as possible".

The pattern proposed in the paper is noted below. It consists of 35 passes, with passes 5-31 being done in a random order:

Pass	Sequence
1	Random
2	Random
3	Random
4	Random
5	0x55
6	0xAA
7	0x92 0x49 0x24
8	0x49 0x24 0x92
9	0x24 0x92 0x49
10	0x00
11	0x11
12	0x22
13	0x33
14	0x44
15	0x55
16	0x66
17	0x77
18	0x88
19	0x99
20	0xAA
21	0xBB
22	0xCC
23	0xDD

24	0xEE
25	0xFF
26	0x92 0x49 0x24
27	0x49 0x24 0x92
28	0x24 0x92 0x49
29	0x6D 0xB6 0xDB
30	0xB6 0xDB 0x6D
31	0xDB 0x6D 0xB6
32	Random
33	Random
34	Random
35	Random

While this method is useful for magnetic media such as hard disks, there is little proof of its effectiveness in solid state media, and, if Wikipedia is to be believed, the method has a 70%+ failure rate on a USB flash drive. These aren't good stats! Gutmann himself mentions the reason for the pitiful performance of his method with modern media in the epilogue to his paper:

"SSDs are a totally different technology than magnetic media, and require totally different deletion techniques. In particular you need to be able to bypass the flash translation layer and directly clear the flash blocks. In the absence of this ability, the best you can hope to do is thrash the wear-levelling to the point where as much of the data as possible gets overwritten, but you can't rely on any given piece of data being replaced, which means that an attacker who can bypass the translation layer can recover the original data."

Essentially, this means that the location of writes in most solid state devices is abstracted. When you overwrite a file, the device may be as likely to just remap the sector you overwrote to somewhere else in the device. The only solution is to write as many times as possible in order to force the wear-leveling mechanism, that is, the mechanism that distributes writes across the drive in order to avoid disk failures, to overwrite the location you desire.

This data destruction nightmare is in part why we previously recommended the use of a ramdisk. Eventually, important data has to end up somewhere where its destruction will be difficult. This is why it's better that it is encrypted when possible.

Secure Erase

Modern SSDs have a function called “Secure Erase”, which is supposed to do what it says on the tin. It fills the disk with zeroes (also called “zeroing out” a device). This feature can be accesses in some BIOS/UEFI software. Secure Erase has superseded most previous government standards for data shredding, and is generally considered reliable. However, there is the remote possibility of a consumer-level implementation being backdoored in some manner. There is also the fact that a lot of BIOS/UEFI menus do not ship with the ability to trigger the process.

The method is also unwieldy, because it requires the erasure of the entire drive. Generally, we would prefer to selectively erase certain files, instead of indiscriminately zeroing out a device. It also does not work for removable drives.

Overall, Secure Erase should preferably be used before carrying out the physical destruction of a drive.

CSEC ITSG-06

The CSEC ITSG-06 is the preferred shredding method of the Royal Canadian Mounted Police, only standing behind Secure Erase. We have chosen to include it because it’s one of the few standards with a level of complexity beyond random writes or zeroing.

The pattern is as follows:

Pass	Sequence
1	0x00
2	0x01
3	Random

Each write in the last pass should be verified. This method is simple, but it is sufficient.

The problem of randomness

The reader has likely noticed that both Gutmann and CSEC rely on randomness. Now ask yourself: what *is* randomness? A computer cannot produce a truly random number, and so, this becomes a problem for us.

Pseudo-random number generators use a “seed number”, an origin, to calculate the final result. This seed is often derived from the system timestamp, which is used for time-keeping (essentially, it lets the computer

know the time). So, the two obvious vectors of attack in order to lessen the effectiveness of data erasure are the manipulation of the timestamp and the manipulation of the random number generation algorithm.

First, let us deliberate on the seed, and on the feasibility of a seed-based attack. Locking a computer's timestamp so you can predict randomness is easy, having the user not notice such a change is hard. To run this attack without it being obvious, the attacker would have to find a way to get the system to report an incorrect timestamp to a single program. This is possible, however, it requires that you have a separate program acting as an intermediate, which intercepts date/time requests and sends back the adulterated timestamp.

Second, there is the possibility that the random number generation algorithm is manipulated in some way. This is also possible, but, if an OS distributor were to adulterate one of the simple algorithms used by most programs, developers would likely detect it, because a backdoor would massively add onto the complexity of the algorithm, leading to spikes in execution time. Such a backdoor would have to be introduced in the implementation of the shredding software. In conclusion, this attack is, in reality, extremely unlikely, and nearly impossible if you implement the shredding software yourself.

If you are writing an implementation, however, something that we definitely do not recommend is using proprietary cryptography libraries, such as Microsoft Bcrypt, to generate random numbers. While the randomness we need for erasing files doesn't have to be cryptographically sufficient, it's still a bad practice to use an obviously backdoored random number generation library.

A readily-available erasure method

If you seek a readily-available data erasure method that you do not have to implement yourself, `shred` is the best one within GNU/Linux. An example usage of `shred` in the command line is as follows:

```
shred -f -u -n 5 file.txt
```

This will erase the file "file.txt", forcing a change in permissions if necessary, deleting the file upon completion, and using 5 passes of random patterns.

For Windows devices, we recommend the freeware tool "File Shredder", which can be found at https://www.fileshredder.org/.

Physical destruction

One way to truly erase data is to physically destroy the storage device it is on. The process of doing this is obvious enough that a child can do it. But we shall go over the most common ways to destroy the most common storage devices.

- **Hard Disk Drives**: Use an extremely strong magnet. Hold it against the device for at least a minute. Hit the device against a wall and crush it. If the destruction doesn't look sufficient, burn the device.
- **Solid State Drives**: Run Secure Erase, and crush it. If the destruction doesn't look sufficient, burn the device.
- **USB Flash Drive**: Remove the drive from its casing. Burn the device.
- **Secure Digital (SD) card**: Burn the device.
- **Compact Disk or other optical media**: Take a fork or a knife to the reflective surface, scrape it multiple times. Burn the reflective surface.

Foreword

In this part of the text, we have introduced methods of secure data removal.

These are relatively simple methods, that any competent work group can easily implement as they see fit. While this may all seem overly technical, it barely scrapes the surface of the subject matter.

BRIEF SECURITY NOTE ON MASS ORGANIZATIONS AND PUBLICATIONS

Moving into the purview of the more conventional day-to-day life of an organization, we shall briefly mention some tools and tips which are useful for both mass organizations and the communist press.

Advice for mass organizations

It is common practice that, for planning labor action, organizations rely on physical meetings. We do not need to harp on the basics: no phones turned on, and no tattling or relaying information via the internet.

However, we also recommend the use of a microphone jammer. Any one of them will do, and if you do not trust store-bought ones, you may opt to build one yourself using one of the many readily-available guides on the internet.

If a mass organization is connected to an advanced grouping, the leadership of the former should adhere to the communication practices of the latter, as to ensure a seamless transmission of information. This means

including the leadership of the mass organization onto the PGP trust chain of the advanced organization's leadership.

Maintaining the security of high-value information is also important, however, due to the lack of centralization inherent in mass groupings, treating such information with the same care as suggested in the former section will be significantly harder. Generally, you can get away with maintaining paper records for most things, which are a step above unencrypted digitally stored information. Delete files you don't need, write stuff down (or use a typewriter!) when possible, and keep in mind that a good lock and a safe are the best forms of mitigation you'll be able to apply.

Advice for the communist press

The security requirements of publications depend on which kind of publication is being dealt with. News publications (not ones that merely analyze current news under a communist lens) are often valued by the ill-intentioned, because they provide an access point to possibly confidential sources and privileged information. We will not tell you about how you should communicate with sources, because that would warrant a good amount of research and a document double the length of this one. If you do not have much contact with confidential sources, we recommend paper records: if the source of the information is an interview, use written records.

Prefer physical meetings for interviews with important sources. If those are not a possibility, do not use common video calling software. As most publications are completely broke, self-hosted video chatting is not a possibility. We find that the most trustworthy browser-based video chatting service is Jitsi Online. However, if you have the resources to rent a server with the required bandwidth for hosting video calls, we recommend self-hosted Jitsi.

For all publications, we recommend the old bolshevik practice of forcing authors to adopt either partial of complete pseudonyms.

WEB HOSTING

The most important part of a publication is what it puts out. Since many publications exclusively use web publishing, it is important to make good recommendations on two fronts: hosting and content production. Hosting is any service a publication may use to make its content available to the outside world, and content production is the process of getting writing into a webpage.

Hosting

A method of hosting that has become quite popular in recent times is one that brings content production along with it. The most well-known platforms that offer this kind of service are blogging platforms Medium, Substack, and WordPress.com (which is confusingly named after the *blogging software* they use, WordPress, but is not the same thing). We do not recommend any of these unified platforms because of a pretty big concern: their business model. While some of them may be free, or offer free tiers, this guarantee is not eternal, and utilizing a paid tier may result in drastic pricing fluctuations.

The largest aspect of this concern, however, comes from the possibility of access to articles published being restricted in the future. Medium, which was a pretty big platform not long ago, started restricting access to articles, and only allowing unlimited reads from subscribers to their service.

There is also a second, smaller concern, centered around migrating content from one host to another. If you just have a whole website on disk, it's far easier to change to a new host. Not so much with these unified platforms.

If you are just hosting a static blog, where interactivity is limited to the client-side and there is no need for dynamic interaction with a server, which is about 99% of what web publications actually are, we recommend using one of the many static hosting services out there, most of which are free. The one we use is GitHub Pages, but there are many alternatives, such as Amazon S3 and Cloudfare Pages. If your content becomes unwelcome in one of them, you can just rebase to the other, simple and easy.

In case static hosting doesn't suit your needs, you can also get a fully-featured Virtual Private Server. The main platforms we recommend are Amazon Web Services and Google Cloud, which, due to sharing a duopoly, offer very flexible (and cheap) pricing. You may also opt to host a webpage on your own machine, though you will need to verify if your Internet Service Provider allows you to open ports 80 and 443 on your router for outgoing traffic, which are required for running a secure web server.

CONTENT GENERATION AND THE COMMUNIST PERSPECTIVE

Availability and page sizes

We must sidetrack ourselves a little to address the main concern of the communist press when it comes to publishing content on the internet, and that is making it readable to as many people as possible across the world.

Consider someone is on the other side of the world, utilizing a limited data plan, on some ancient computer in a public internet café. Many people

browse the internet like this, and for them to read our work, we need our pages to look inconspicuous from the next chair over, we need them to load up quickly, and we need them to take very little space.

Aesthetic considerations must be made completely secondary to actual functionality. However, they must be primary specifically in the case of readability. Typography, spacing, and framing are all important in the lexicon, and they avoid a web page being placed in the hall of fame of “marxist web design”. The text must be pleasant to read and the page sizes must be small.

Rising page sizes are a worrying trend all-around. The HTTP Archive, which is the authoritative source when it comes to calculating the average size of web pages across time, points at a total growth of 427.6% in the average size of desktop web pages from late 2010 to late 2023. It has risen from 467.7 Kilobytes (KB) to 2467.5 KB, with the current worst-case scenarios being around 5000 KB.

This can be squarely blamed on the trend towards flashier web-pages, with more animations, images, and, of course, advertising. In fact, if we take advertising into account and start talking in terms of total page sizes beyond an initial capture, some web pages have a total page size of $\mathbb{R}$ (yes, the entire realm of the real numbers), as constantly refreshing ads add onto it infinitely. In summary, this trend of bloat and inaccessibility is a bourgeois trend, which communists must take no part in.

Let’s engage in a practical example. What if we took a sizable work in plain-text, and set an acceptable range for the size of its paginated version at 5 times the size of the plain-text version?

There is no better work to do this with than with the English Edition of the Manifesto of the Communist Party. As it is a relatively large text, and is completely in ASCII (every character is one byte). The plain-text version distributed by Project Gutenberg, excluding the header and footer, is around 72.5 KB in size, this means that, at 5 times the size, our paginated version would sit at around 362.5 KB, lower than the average desktop page size in 2011!

Due to the existence of browser caching, a larger size is acceptable for individual webpages. Our recommendation is that every individual webpage should attempt to maintain a total file-size (total, in this case, meaning the size of all data received by the client-side when opening a webpage, not just the file-size of the markup file) below 512KB, with an area of tolerance between 512-768KB.

Content generation

There are many ways to turn a piece of text into a formatted webpage. We shall name some tools, and then give a practical example with the simplest possible tool you may use.

The *blogging software*, WordPress, doesn't produce a collection of files that can be statically hosted, but it's worth mentioning, as it is extremely scaleable and is probably the best fit for large outfits that need to put out content quickly. Its blogging model, with posting permissions able to be delegated between many editors, is extremely useful in that regard. The software also has extensive support for custom themes and plugins, however, it should be noted that user error in setting up such plugins may leave ample room for exploits by bad actors, and a badly-developed plugin may introduce serious vulnerabilities. There is also the problem of page sizes. Wordpress pages being some of the most bloated all-around, with a "small" Wordpress site being around 2000KB.

Wix, one of the "all-encompassing" website creation platforms, has many articles about it which allege that websites made on it can be exported, however, we haven't been able to find that option during our own testing. While most Wix websites are on the high-end when it comes to size, it is perfectly possible to create a lightweight website with very little effort (we have seen one at 140KB!). The platform, however, is a bit of a pain to work with, as its mainly directed towards the creation of portfolios and shop pages.

RocketCake is a great static website editor, which we have used extensively before to create many pages. It is a great tool all-around, and it is easy to create lightweight webpages in. However, it's a bit clunky to use in a blogging workflow. If you're not willing to shell out money for the professional version, you'll have to edit the generated pages yourself to add custom HTML or Javascript, which is generally a pain.

The bare-bones content generation pipeline

The simplest possible workflow does not involve specialized website creation tools at all, but a text editor and a tool to convert Markdown, a human-readable markup format with a dead-simple syntax, into a formatted HTML document.

Any text editor will suffice. The tool we will use for the conversion is Pandoc, the state-of-the-art text of text format conversion software.

Let us take this example Markdown:

```
# Big text
## Smaller text
```

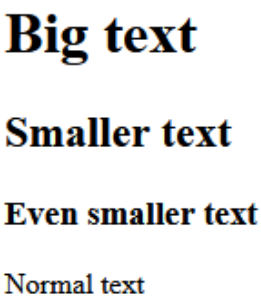
```
### Even smaller text

Normal text
```

Now, suppose this is saved to a file called “text.md”, if we wanted to convert it to HTML markup saved to the file “text.html”, we would use the following syntax with Pandoc:

```
pandoc -s -f markdown -t html5 -o text.html text.md
```

Where -s tells pandoc that we want to save the output to a standalone file, -f specifies the origin format, -t specifies the target format, -o specifies the output name, and “text.md” is the input file.

Once we run the command, and open the HTML file in a browser window, we get something that looks like this:



A great start! And, if you wanted to create an extremely simple series of webpages, linking one page to another, this would suffice. However, most people want something more than that. For this, you’ll have to learn HTML and CSS, which this document won’t cover. If you want a good resource to learn, W3Schools is pretty much the place, with tons of free tutorials and an interactive editor.

If you know HTML and CSS, you’ll want to take a look at the file generated by Pandoc. you may notice that you’ll have to do the work of separating the text into divs yourself, and that you can easily define the styles which will be used for things like code blocks. You may also create a style.css file, which you can include in your Pandoc export via the -c command argument.

Pictures

A group named Firebrand published an article, “Protest Photos: Facing the Unblurred Truth“, which served as a major inspiration for this whole text, and perfectly embodies the neanderthalian beliefs that are ripe among organizations which refuse to advance their political culture beyond the 20th century.

In response to suggestions that the faces of protesters should be blurred before photos are published, they say “[...] we cannot let individual safety concerns solely dictate the path toward our liberation”, arguing that “In removing or suppressing protest photos, we risk losing control over our narrative, allowing the story to be manipulated by those with dubious intentions, driven by bourgeois ideology”. Firebrand clearly assumes the position that compromising the safety of protesters is part of an effort to “control our narrative”.

They say, “disguising identities will not shield us from a dystopian future”, we say that this is not a “dystopian future”, but a common reality of today. They say “safety and protection comes from the strength of unity within mass liberatory movements”, we say that such a thing is not birthed from magic, and that no amount of “people’s power” can protect someone from comparisons against a database of faces, or against attempts at geolocation.

We will take the debate out of the domain of rethoric and into the domain of science by providing a sample of what can be done with a simple picture, without bothering about the contents of the image at all, and merely peeking at the data.

EXIF data

Let us begin our example with a short explanation. Most pictures taken on smartphones today contain GPS information and other data about the photographic device, using a metadata format called EXIF (Exchangeable Image File Format).

This poses an obvious security issue, and it has been treated as such. Most social media and blogging platforms today remove EXIF data from published images. However, this is not a guarantee that they aren’t keeping the data for themselves. This problem extends beyond major platforms and into self-hosted ones. For example, in Wordpress, metadata removal in full-sized images is handled by third-party plugins, which means an attacker could introduce a compromised plugin to extract and store EXIF data.

But what metadata may a picture contain, exactly? That depends on the device it’s taken on, but for presentation purposes, let us take an image from

the git repository located at https://github.com/exif-samples, at `jpg/gps/DSCN0010.jpg`.

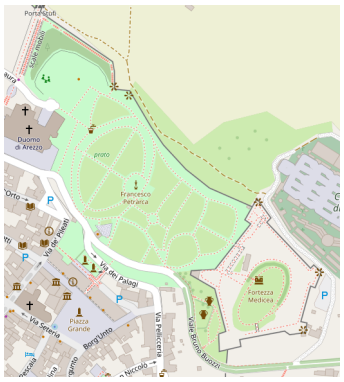
A nice picture. Now, let's open a terminal window and inspect it with `exiftool`, a program for examining modifying image metadata on Unix-like machines:

```
exiftool DSCN0010.jpg
```

The output is long, presented in plain-text, and contains more metadata than the one contained in the EXIF data itself, but are the critical parts, compiled into a spreadsheet for ease-of-viewing:

GPS Latitude Ref	North
GPS Longitude Ref	East
GPS Altitude Ref	Above Sea Level
GPS Time Stamp	14:27:07.24
GPS Satellites	06
GPS Map Datum	WGS-84
GPS Date Stamp	2008:10:23
GPS Date/Time	2008:10:23 14:27:07.24Z
GPS Latitude	43 deg 28' 2.81" N
GPS Longitude	11 deg 53' 6.46" E
Circle of Confusion	0.006 mm
Field of View	18.3 deg
GPS Position	43 deg 28' 2.81" N, 11 deg 53' 6.46" E

Not good! Using this information, we can pinpoint this image to the northern end of Parco della Fortezza Medicea, Arezzo, Tuscany, Italy. And if we actually put in the effort, we could probably figure out the exact angle it was taken from.



Thankfully, we can remove all this undesired data with a simple command in `exiftool`:

```
exiftool -all=image.png
```

Now, this image has no compromising metadata, and can be safely published. If we were to re-examine the metadata after running this command, we would find nothing useful.

Avoiding facial recognition and foreword on pictures

Avoiding facial recognition is simple: whatever Firebrand says, you do the opposite of. Do not show faces of fellow communists, unless they explicitly consent, and do not pressure them to consent. Many social media websites use facial recognition tools, and the existence of the image opens up a massive digital footprint which can be exploited by bad actors, the problem with posting exploitable information to the internet is that its lifetime is indefinite, and so the risk is forever-lasting.

If you are frequenting a protest, wear a mask. Now that hygienic face masks are normalized, this is trivial to do without looking suspicious. The most important area to be covered is from the nose downwards, but, during a sunny day, you can use sunglasses.

As for other public events, if those are centrally organized, banning the taking of pictures by organization members is good practice. This is a scenario where no one can escape the cameras and the microphones, but avoiding them when possible is also a good idea.

Beyond practical considerations, we must finish addressing the deficient concept that creates the negative security culture perpetrated by organizations such as Firebrand: The pride around the figure of the public individual revolutionary. This is a misinformed perception which reeks of historical reenactment.

The public revolutionary, which linearly builds up an outspoken presence, was a product of their own time, of a period where the capitalist State did not have overwhelming control over communications, which it uses in a constant and automated manner in order to maintain social peace. Today, the situation is completely different, and building up the image of a single orator can be dangerous and outline them as a political target at a premature stage. This does not mean we have to do away with orators, it just mean that they cannot become the center of an organization's political image. Not only does it lead to idol-worship, it's also bad tactics.

We must also recognize another insidious reasoning for the negative security culture, and that is the fact that most of its individual proponents hope to become the public orators of the future. Independent of their reasoning, be it a careerist reasoning, or one based around the more noble cause of representing and speaking for their specific communities, this is an individualistic position at its core, and a position that ignores the reality of the situation for the purposes of creating more "great men".

The communist position should be a scientific one. We hope that all the recommendations which we have presented in this text convince the reader of the necessity of a positive culture around security.

FOREWORD

It is certainly difficult for an organization to put into practice the overview we have given in this text and to create guidelines around it. Few organizations have anyone well-versed in both political and technical matters which can be tasked with directing an important facet of internal policy.

However, this doesn't mean that adapting policies to the digital present is something that should be ignored. There are always people with the will and skills, but often, bureaucratic formalism prevents them from reaching their full potential within an organization. The concept of work-groups in general is great to minimize these issues, allowing the merger of technical and political knowledge.

Keep in mind, this text is merely a primer, introducing basic notions and allowing for further research. Topics we have brought up here, such as content production and accessibility, are certain to have more written by us on them in the future. Specifically, we plan to one day author a piece on an accessible content (as opposed to form)-driven web.

In the future, it would be wise to open a debate on creating a modern communist media culture, that exists outside of mass media, instead of subjugated to it. We are currently making efforts towards that debate, in a practical sense. We should also look at the role of decentralization beyond the petty-bourgeois fantasy. There is a practical effort towards this, as well.