Naeon: A method for secure distribution of confidential data to an untrusted environment

René Smaal

rsmaal@naeon.nl

Abstract: The global datasphere has grown faster than we could imagine a decade or so ago, and will keep doing so in the years to come, at a rate we may have a hard time imagining today. In its report *Data Age 2025*[1], the IDC stated that in the year 2025 the annual data generated worldwide will increase to 163ZB from 33ZB in 2018, which is equivalent to 446EB of data per day[2]. Keeping control of this dazzling amount of data, and keeping it safe from both internal and external threats, raises a lot of challenges. This new digital landscape we now find ourselves in, where an exponential growth of data goes hand in hand with an ever growing threat of cyber attacks and data breaches, demands new techniques and ways of working to keep control over our data. Storing data 'in the cloud' solves at least the problem of data loss by physical causes like fire, flood, earthquake, or hardware malfunction. But it also introduces many new problems and challenges. In the case of commercial cloud providers for example, you can never be sure if your data is stored safely and securely. Think data leakage, data abuse, privacy snooping. A third party commercial cloud provider can shut your account down and take away all data at their whim, for whatever reason, rightful or not. They can decide to stop their services as they see fit. Or even go out of business altogether. So if integrity, security, confidentiality and long-term availability of your data is of any concern, entrusting them to a commercial cloud provider may not be your safest bet. This work aims to be a method, as well as a tool, for secure distribution of confidential data to any off-site storage environment, in such a way that its confidentiality, integrity and availability remain guaranteed at all times, regardless of the trustworthiness and reliability of the chosen environment. In addition, it includes a quick, explorative sketch of a robust cloud platform that combines long-term availability with rigid security techniques, while at the same time being completely independent of a single authority.

Index Terms: access control, cloud storage, content-addressed storage, privacy, secure data access, symmetric key cryptography

I Introduction

In December 2020, in the midst of the corona pandemic and subsequent lockdowns, cyber-criminals worldwide had begun to take shameless advantage of the situation by launching ransomware, phishing, vishing, and all the other attacks they could find in their trick boxes, aimed at security holes in the IT infrastructure of their targets. In the Netherlands, cyber attacks — more specifically ransomware attacks — started gaining public attention when institutions belonging to the crucial infrastructure, such as *The Maastricht University of Netherlands* and the *Overijssel municipality of Hof van Twente* were successfully targeted by ransomware attacks. And at the time of writing of this paper, *Kia Motors America* suffered a ransomware attack, and both the *Amsterdam University of Applied Sciences* (AUAS) and the *University of Amsterdam* (UvA) were hit by an unspecified cyber attack. In the December article *The Worst Hacks of 2020, a Surreal*

Pandemic Year[3], Lily Hay Newman of Wired reported cyber attacks on SolarWinds, Twitter, Blueleaks, University Hospital Düsseldorf, Vastaamo and Garwin, all but one carried out in the second half of 2020. Datto, a provider of white-label cloud services for managed service providers (MSPs), published its global survey *Datto's 2020 Global State of the Channel Ransomware Report,* based on the findings of 1,000 MSPs on the state of the cybersecurity landscape. The survey pointed out that small and midsize businesses (SMBs) remain top targets for ransomware attacks: nearly 70% of MSPs report SMBs have been hit with ransomware in 2020. And "while ransoms are pricey (~$5K), the real cost is in downtime and loss of productivity due to the time-consuming recovery that often follows an attack." According to MSPs, 62% of their clients' productivity was impacted by the attacks, and 39% mentioned that their clients even suffered from business-threatening downtime as a result[4]. The stories of honest, hardworking enterpreneurs who, after falling victim to cyber criminality, were forced to shutdown their businesses and dismiss their loyal employees, are no less than heartwrenching. This may be just the tip of the iceberg, as there are no numbers on how many organizations that fell victim to said crimes opted to remain silent due to the fear of substantial reputational damage. There is no shortage of specialists in the field of cybercrime prevention, nor is there one in the field of damage control after the fact. Yet, despite of this, statistics keep showing a staggering increase of damage caused by data loss as a result of cybercrime. Prevention is better than cure, as the saying goes. But no matter what prevention measures are put in place, no matter what security policies are implemented, loss of critical data due to cybercrime (or any cause, for that matter) can never be completely ruled out. And when prevention fails, the cure may prove costly or, worse, even impossible. If prevention is not always the answer, and a cure not always attainable, what solution is there to think of, when our goal is to address the dangers of cyber attacks proactively? By the end of December 2020 we started a conceptualization process that, by mid-February, had resulted in a working proof of concept of such a solution. This proof of concept is, in essence, a safety net for exactly this type of situations, where both cybercrime prevention and -cure are out of reach. It is designed with the single purpose of ensuring the security and privacy of the data it processes by using some of the strongest cryptographic algorithms currently available, so that only you yourself will ever have access to it[5]. Additionally, a sharding technique is applied, whereby the encrypted data is split up into a random number of chunks that are similar in size, time-stamped equally, and with hashed filenames that do not reveal any information about their content, nor about their concatenation order. Without knowledge of the latter, the number of possible permutations alone is such[6] that even though all the chunks get stolen or compromised in any way or form, there is no way the attacker could successfully concatenate them into their original order to form a valid, decryptable file.

The rest of this paper is structured as follows. Section II is about *Naeon*, a data encryption and sharding method. Section III contains a rough sketch of the outlines of *Naeon Pool*, a blockchain-based, cooperative distributed cloud environment that is tailormade to host the output (chunks) of *Naeon* in such a way that long term availability, as well as quick, secure, reliable and easy access are guaranteed at all times, without being dependent of a single authority. Section IV contains a case study. It ends with a conclusion in Section V.

II NAEON

In order to reduce the chances of data loss as a result of a cyber attack, backing up your data off-site is one of the primary foundations of good disaster mitigation policies. The term "off-site" in the context of data backups has become synonym with "the cloud" in recent years, and with it, the type of risks involved changed drastically. The traditional, physical, off-site data storage required no more than a trusted location, a safe environment. The cloud, on the other hand, is a virtual

space in which the (potential) dangers looming are far more complex, impossible to predict, mostly unknown, and highly dynamic in nature. Once you upload your critical data to the cloud, you lose control over it. From that moment on, your data is in the hands of the chosen cloud storage service, and you can only hope that they have their security policies in check, and first and foremost that the service itself is to be trusted at all times. Even if you encrypted your data before entrusting it to a cloud service, there is always a possibility that one overlooked vulnerability exposes your data to prying eyes. It is therefore essential that you take the best possible security measures before you entrust your data to a cloud service, or any off-site storage for that matter.

Naeon is a working proof of concept, implemented as a free and Open Source Bash program under the terms of the GNU Affero General Public License as published by the Free Software Foundation[7]. It addresses potential security threats to which all data are exposed *before and after* 'they leave the building' on the following levels:

1. **Confidentiality**
   Confidentiality refers to the protection of data from unauthorized access or disclosure. The *International Organization for Standardization* (ISO) defines it as: "property that information is not made available or disclosed to unauthorized individuals, entities, or processes."[8]

2. **Integrity**
   Integrity refers to the protection of data from unauthorized modification or destruction, or by the definition of the ISO: "property of accuracy and completeness."[9] Data should be consistent as well as trustworthy over its entire life cycle, without it being altered in transit by unauthorized individuals.

3. **Availability**
   Availability refers to the protection of data from unauthorized disruption. The ISO defines it as: "property of being accessible and usable on demand by an authorized entity."[10] It assures that your data can be accessed (only) by authenticated individuals whenever needed.

The following methods and techniques are used by *Naeon* in order to meet the highest possible standards of information security management:

1. **Rijndael block cipher**
   Naeon uses the Advanced Encryption Standard (AES) 256-bit encryption algorithm — also known by its original name Rijndael — in combination with a 128 character passphrase, randomly generated from the alphameric set, generating a 256-bit encryption key which has 2^{256} possible combinations. An attacker would have to try most of these 2^{256} possible combinations before getting it right. This would take way beyond any human lifespan[11].

2. **Obfuscation**
   Standard symmetric encryption algorithms result into an encrypted file with a particular format, which contains the encryption key, and which can be identified, by the same algorithm or any other suitable decryption tool, as a valid, decryptable file by checking its properties, or by simply performing a decryption attempt. Both the fact that the encrypted file contains the encryption key, and the fact that the file can easily be identified as an

encrypted file without ample effort, are potential vulnerabilities. Knowing that it concerns an encrypted file, or even worse, which algorithm was used for its encryption, is information that attackers shouldn't be able to distil to begin with. Furthermore, storing the encryption key in an encrypted file that can be recognized as such, opens the possibility of a successful brute force attack. *Naeon* addresses both issues using the following two methods:

Making the encrypted file unidentifiable

Two blocks of random bytes are generated. The first block, the so called *prepend block*, randomly sized between 1 and 10,000 bytes that are indistinguishable from ciphertext, is prepended to the encrypted file, so that the encrypted file now starts with random bytes, instead of its original header. The second block, the *append block*, is then appended to the encrypted file. Just like the prepend block, it consists of random bytes, but other than the prepend block, its size is not random but calculated, so that the newly formed container (prepend block + original encrypted file + append block) can be split into equal parts of 10n bytes each.

Preventing the encryption key to be sent out with the encrypted data

The container, which now encapsulates the original encrypted data plus the prepend- and append blocks, is then split into a private chunk, and *c* equally sized public chunks of 10n bytes, where $100 \leq c < 1000$, and *n* is an integer where $n \geq 2$. The private chunk contains the prepend block, the original header of the encrypted file *including the encryption key*, and the first part of encrypted data, up to the point where the first public chunk starts. The public chunks contain the rest of the encrypted data (the last public chunk also contains the append block) and are meant to be sent out. The private chunk stays with the user, and needs to be subject to proper key management.

3. **Sharding and generalisation**
   The container is sharded into one private chunk and 100 to 1,000 equally sized public chunks. Each chunk, both the private and the public chunks, are then renamed with their SHA-512 hash values. A filename conversion table is created with the original chunk filenames (containing their concatenation order) and their corresponding SHA-512 hashes, so that the concatenation order can be restored during a future restore. The second generalisation step is to timestamp all chunks with the same time: January 1st 2020, midnight UTC. Now all public chunks are equally sized, have the same timestamp, and are named in a way (their hash value) that doesn't reveal any information about their content, or about their concatenation order. Without knowledge about the concatenation order, concatenating 100 to 1,000 chunks in the right order is virtually impossible[6]. Non-concatenated, individual public chunks, being isolated from their original context of an encryption file format, are no valid encrypted files, can therefore not be decrypted, with or without a key, and as a result are useless for any attacker.

An example Naeon backup flowchart is shown in Figure 1. Figure 2 contains a sketch of the Naeon backup- and restore protocol. A partial listing of the Naeon Master Key XML is shown in Figure 3. Figure 4 shows a sample output of Naeon in backup mode, securing all existing MySQL databases by first performing an ASCII dump.

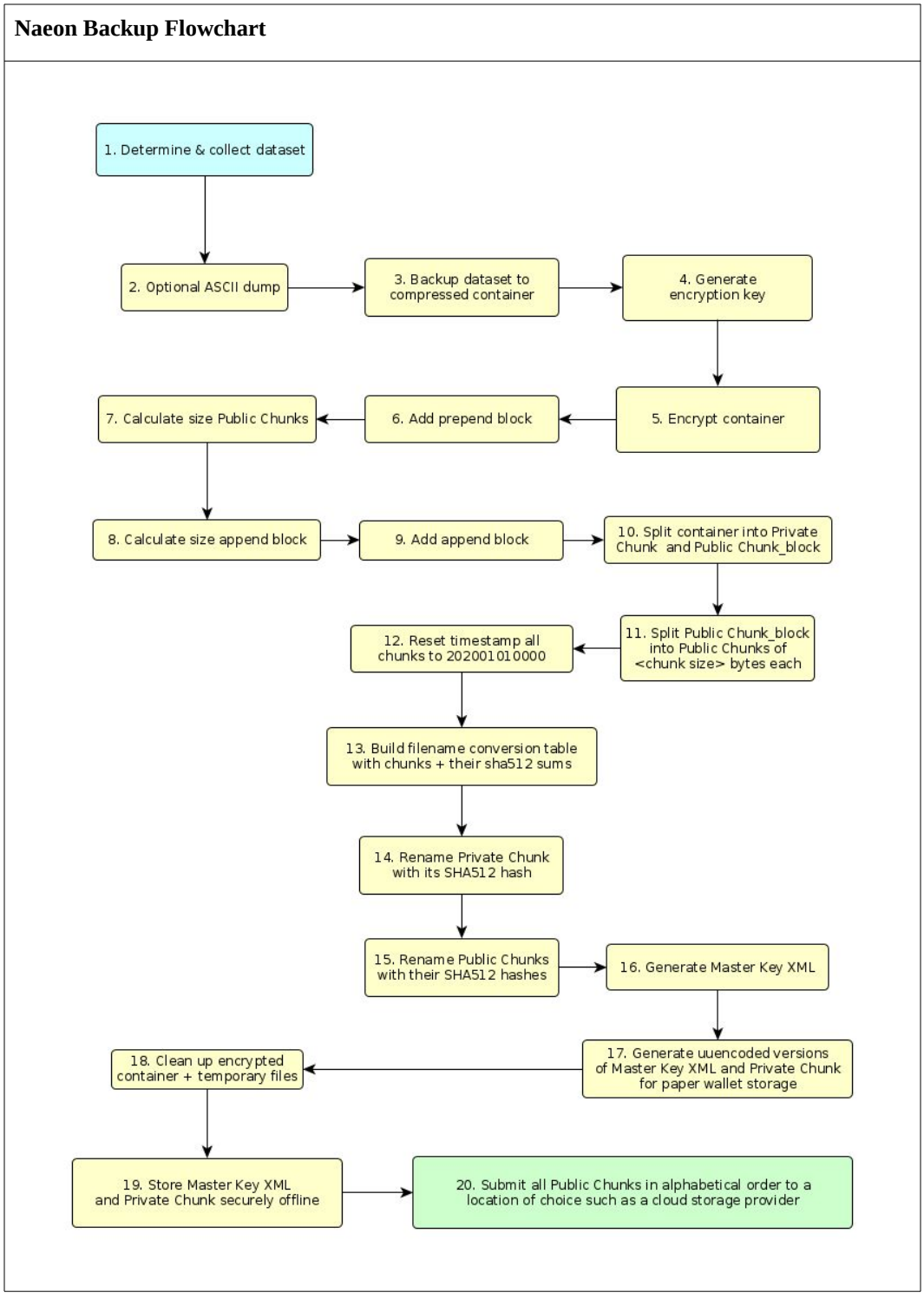

Figure 1: Naeon backup flowchart

Naeon Protocol Sketch

Backup

1. Create a Naeon project directory "naeon-<yyyymmddhhmmss>" with one private- and one public subdirectory
2. Collect input files
3. Compress input files into compressed archive
4. If size of compressed archive exceeds minimum of 100K, continue
5. Generate a random 128 character passphrase from the alphameric set
6. Encrypt compressed archive with generated passphrase using aes256 encryption
7. Generate a random sized prepend block between 1 byte and 10K bytes and prepend it to the compressed archive
8. Calculate the number and the size of the public chunks (c) that the encrypted archive will be split into: $c \times 10n$ bytes, where $100 \leq c < 1000$, and n is an integer where $n \geq 2$.
9. Add an append block of *n* bytes to the encrypted archive so that the size of the last chunk equals that of all the preceding chunks
10. Apply sharding to the encrypted archive by splitting it up into one private and (c) equally sized public chunks
11. Mark the first chunk as private, and the rest of the chunks as public chunks
12. Reset the timestamp of all chunks to January 1st 2020, midnight UTC
13. Create a filename conversion table with the original chunk filenames (containing their concatenation order) and their SHA512 hashes
14. Rename all (both private and public) chunks with their SHA512 hash values
15. Move the private chunk into private directory
16. Move the public chunks into public directory
17. Create a Master Key XML in the private directory, containing all relevant metadata required for a successful future restore
18. Create plaintext uuencoded versions of the private chunk and the Master Key XML for paper wallet usage
19. Create a filelist of all backed up files and save it in the private directory
20. Clean up all temporary files and directories
21. Log eventual errors in an error log in the private directory

Restore

1. For each Naeon project ID to be restored:
2. Check if both a private and a public Naeon directory exist
3. Check if the private directory contains a private chunk, a public chunk, and a Master Key XML
4. Check if the public directory contains the number of public chunks in accordance with the number mentioned in Master Key XML
5. Perform integrity check on private and public chunks by comparing their SHA512 values to their filenames
6. Copy all chunks to a temporary directory with their original filenames (with concatenation order) using the filename conversion table in the Master Key XML
7. Concatenate the chunks into their original order, thereby recreating the original container
8. Remove the prepend block from the container
9. Remove the append block from the container
10. Decrypt the container using the passphrase from the Master Key XML
11. Decompress the decrypted compressed container (unless the option –nodecompress was selected)
12. Clean up all temporary files and directories

Figure 2: Sketch of the Naeon Protocol

Naeon Master Key XML

<?xml version="1.0" encoding="UTF-8"?>

<naeon>

<Naeon_version>1.0</Naeon_version>

<Issuer>Naeon</Issuer>

<Issuer_notes>For demonstration purposes</Issuer_notes>

<Filename>naeon-backup-20210221092245.tar.gz</Filename>

<sha512hash>fded2d6afc9094f316fb597b31b217905aafbf9ac30356b9d6c030038baba3e58084
2417ed92c01c529b553d378d36af7487be47b93e5aac5e09efe0d0aa20fe</sha512hash>

<Description>Naeon backup MySQL dump all databases</Description>

<Local_timezone>CET +0100</Local_timezone>

<Backup_timestamp>20210221092245</Backup_timestamp>

<Backup_ID>20210221092245-
DkuAwmsvbvSdjaYKrPTSRwygRoihDkeNknULTMrRZHFqwbjZoRccWzPjmJeyfmrv</
Backup_ID>

<Embargo_timestamp>20200101000000</Embargo_timestamp>

<Selfdestruct_timestamp>99991231235959</Selfdestruct_timestamp>

<Delete_on_retrieval>no</Delete_on_retrieval>

<Encryption_program>gpg (GnuPG) 2.2.27</Encryption_program>

<Encryption_command>gpg --cipher-algo aes256 -c --for-your-eyes-only --batch --pinentry-
mode=loopback --passphrase</Encryption_command>

<Passphrase>tWzCVN0gDL3llhKTwaUiUsBD2qnyrc5zdoXtxoNWkXhQBe3lbezouS71W2Q7
CjNTYhbUGg8XFG1ZpvWFzOEKmLjr8mMUAWATeUlvMdfPoQQGBpPFtkRay5knEkUjfr
Ib</Passphrase>

<Prepend_bytes>9954</Prepend_bytes>

<Append_bytes>358</Append_bytes>

<Number_of_public_chunks>448</Number_of_public_chunks>

<Extension_of_public_chunks>.naeon</Extension_of_public_chunks>

<Location_of_public_chunks></Location_of_public_chunks>

<PriChunk>35a75185d37a7df4737f87195909944aacfb907b8adc7e6f71f62f098ddb6b1ad70deb
ad06852844040e2fea413c519d49d91aad991610aaf4a80df468e64672</PriChunk>

<PubChunk>43c1f67c60c64b79076c362c3a80b94109ff3b04ab03b58f19055238779ceffa7d4bd
b934ed011591f0b73c72b92a13b8234b72d82cb0fcdaaaef694bada6c23 xaaa</PubChunk>

...

<PubChunk>d192d299717c8850c41584058d6744943666cade6bc3c717732ab7eefb64e213eaee
cc77f349112de711c50224f00be332a5392b2e1455805927609e4ceb234b xare</PubChunk>

<PubChunk>cadacce899ff676e5f99cc7fe79800cf0721149b70634e00b07181ea442eefc0b7167b
1ebcc7400f71c9467990b962d666b628eea0104312cb52c74bb60e190d xarf</PubChunk>

</naeon>

Figure 3: Example of a Naeon Master Key XML

Naeon Backup Sample Output

```
naeon > mysqldump --all-databases --single-transaction --quick --lock-tables=true  -u rene > full-backup-mysql-$(date +%Y%m%d%H%M%S).sql
naeon > ls -l
total 476
-rw-r--r-- 1 rene rene 487019 Feb 21 10:17 full-backup-mysql-20210221101752.sql
naeon > 

naeon > ./naeon -b full-backup-mysql-20210221101752.sql
Initializing...
```

```
Version 1.0
This is free software with ABSOLUTELY NO WARRANTY.
Requires: libxml++ 3.2.2-1, bc 1.07.1, coreutils 8.32-1, GnuPG 2.2.26-1, sharutils 4.15.2-4, libseccomp 2.5.1-2.
Environment: Linux
Please use this program on a computer which is disconnected from the internet!
Starting naeon in backup mode.

Press any key to continue

Enter the description of this backup (only alphanumerical characters allowed) :

Naeon backup MySQL dump all databases
Starting backup...
Creating /home/naeon/Program/naeon-20210221092245/naeon-20210221092245-private.
Creating /home/naeon/Program/naeon-20210221092245/naeon-20210221092245-public.
Creating '/home/naeon/Program/naeon-20210221092245/naeon-20210221092245-decompressed'.
Creating '/home/naeon/Program/naeon-20210221092245/naeon-20210221092245-compressed'.
full-backup-mysql-20210221101752.sql
Creating compressed archive 'naeon-backup-20210221092245.tar.gz'. Please wait... Done.
Checking compressed archive 'naeon-backup-20210221092245.tar.gz'. Done.
Generating encryption key..
Encryption key generated. This key will be stored in the tag <Passphrase> in your Master Key XML
Encrypting 'naeon-backup-20210221092245.tar.gz'...
File naeon-backup-20210221092245.tar.gz encrypted to 'naeon-backup-20210221092245.tar.gz.gpg'
File size of 'naeon-backup-20210221092245.tar.gz.gpg' is 487425 bytes.
Generating prepend block of random bytes:
1+0 records in
1+0 records out
9954 bytes (10 kB, 9.7 KiB) copied, 0.000230731 s, 43.1 MB/s
Inserting prepend block of 9,954 bytes into 'naeon-backup-20210221092245.tar.gz.gpg'..
Prepend block inserted.
File size of 'naeon-backup-20210221092245.tar.gz.gpg' is now 497379 bytes.
Calculating chunk sizes..Chunk sizes calculated.
Adding append block to 'naeon-backup-20210221092245.tar.gz.gpg'..
1+0 records in
1+0 records out
358 bytes copied, 0.000189898 s, 1.9 MB/s
Append block added.

497,737 bytes in 'naeon-backup-20210221092245.tar.gz.gpg':
9,954 random bytes prepended + 487,425 bytes encrypted data + 358 random bytes appended, to be distributed as follows:

49,737 bytes in Private Chunk
448,000 bytes in 448 Public Chunks of 1,000 bytes each

Splitting 'naeon-backup-20210221092245.tar.gz.gpg' into private and public chunk block..
Splitup completed.
Generating 448 public chunks of 1000 bytes..
Public chunks generated.
Resetting timestamp of all chunks to Jan 1, 2020..
All chunks timestamped.
Building a name table with original names + their sha512 sums.. Done.
Renaming private chunk with its SHA512 hash and moving it to private directory..
Renaming public chunks with their SHA512 hash values..
Public chunks renamed to their hash values with extension .naeon.
Moving public chunks to public directory.. Done.
Creating the Master Key XML.. Done.
Formatting the Master Key XML.. Done.
Creating uuencoded ASCII versions of Private Chunk and Master Key XML for paper wallet.. Done.
Removing compressed archive 'naeon-backup-20210221092245.tar.gz'.. Done.
Cleaning up temporary files and directories.. Done.

F I N A L L Y

SECURE YOUR PRIVATE KEY: WITHOUT IT, NO ONE CAN RETRIEVE YOUR DATA!

Store the (content of) the directory '/home/naeon/Program/naeon-20210221092245/naeon-20210221092245-private'
(containing your Master Key XML and your Private Chunk) into a safe and secure
place. This can be a digital wallet, or even a (physical) vault if you like.

Important to keep in mind: Your data will be irrecoverably lost in case you lose
access to your private key. The files with the extension .UUE are meant for
usage in combination with a paper wallet.

The safety of your data depends on how secure you keep these files
from others than yourself.

SEND YOUR PUBLIC CHUNKS "OUT TO STAY"

Upload the (content of) the directory '/home/naeon/Program/naeon-20210221092245/naeon-20210221092245-public'
(containing the Public Chunks) to a location of your choice, preferably one that
provides long term availability, as well as quick, secure, reliable and easy
access. It is up to you whether that is a traditional cloud storage provider, or
a blockchain-based cooperative distributed cloud that keeps you independent of a
single authority.
```

Figure 4: Naeon Backup Sample Output

III NAEON POOL

One of the great strengths of the *Naeon* concept when it comes to information security, is the apparently contradictory quality that the bigger the pool of all users' uploaded public chunks, the higher the level of security attained for each individual user's public chunks. This is due to the fact that all uploaded public chunks of all users are indistinguishable from one another. All public chunks are timestamped January 1st 2020, midnight UTC. All are sized 10n bytes (where $n \geq 2$). And all their filenames are equal to their SHA-512 hash values, so that it is virtually impossible[12] that two chunks will bear the same name. These hashes for filenames have no information value whatsoever, except for the keeper of the Master Key XML, who is able to retrieve the original concatenation order, using the filename conversion table it contains. It follows that the higher the number of elements in such a pool where all elements are stirred together into one big blend of homogeneous data, stripped from any context, the smaller the share of each individual user. And, consequently, the harder it will get for an attacker to successfully trace back any element or elements to its original user.[13]

Let's illustrate this with an analogy. A group of people are at the beach. Every person in the group holds a bottle filled with water. The content of each bottle is similar (let's say this concerns water coming from the same source). The only attribute that sets the content of each bottle apart from that of the other bottles, is the bottle that holds it. All bottles are emptied in the sea. Now imagine every bottle has the property of 'knowing' the name of each H_2O molecule it contained, and that it could summon all molecules to immediately return to the bottle once called by their right names. This is analogous to the way *Naeon* handles the backing up and restoring of public chunks. The successful retrieval of public chunks from the pool by an unauthorized individual using brute force techniques will remain purely hypothetical[13], as the chances of guessing their SHA-512 hash values right, are negligible.

Now let's sketch the rough outlines of *Naeon Pool* ('the pool'), an imaginary cloud storage environment that would be ideal for *Naeon* public chunks to be stored in, in terms of information security. The pool is not intended to be a file sharing network, but rather a permanent archiving solution. Ideally, it should have at least the following properties:

1. A distributed, decentralized, self-regulating, node-based information storage and retrieval system, which operates without a central authority, yet under a strict set of rules.

2. Driven by three parties: 1. Submitters of data, 2. Retrievers of data (not necessarily being the same parties)[14], and 3. Data storage providers.

3. Submitters can submit an unlimited amount of chunks to the pool.

4. Retrievers can retrieve an unlimited amount of chunks from the pool.

5. Anyone can be a submitter, retriever, provider, or any combination thereof.

6. The cost for the pool are carried by the submitters and retrievers of the data (chunks) as a proportion to their data usage.

7. The revenues from the pool go to the providers of data storage in proportion to their ability to host data for the pool.

8. The pool consists of 3 layers: an abstraction, an anonymization and an incentive layer.

9. The abstraction layer obscures and conceals all information concerning content (file listings) for all parties.

10. The anonymization layer conceals all information concerning the identities of submitters, retrievers and providers of data, and the transactions between them.

11. The incentive layer regulates the money stream between parties.

12. The administration of the pool (abstraction-, anonymization and incentive layers) is carried out by a blockchain.

13. Chunks can only be retrieved from the pool by requesting their correct filenames (which is their sha512 hash, followed by the extension '.naeon').

Naeon has some distinct features that separate it from other encrypted backup methods, and are potentially very powerful in their possible applications. Since these may be unique to *Naeon*, the *Naeon Pool* should support them as well. It concerns the following two timestamps and a flag that can be attached to any Naeon Project ID, which will then be attached as metadata to all chunk uploads belonging to that Naeon Project ID.

14. **Embargo Timestamp**
The pool will *deny all retrieval requests* before the Embargo Timestamp, if set[15].

15. **Selfdestruct Timestamp**
The pool will *automatically remove all chunks* at the Selfdestruct Timestamp, if set[16].

16. **Delete On Retrieval Flag**
The pool will *automatically remove chunks* once they are successfully retrieved from the pool, if set[17].

A quick comparative survey of decentralized, distributed data storage and retrieval platforms that are currently available, show that much progress has been made in this field in recent years. Of all platforms reviewed, Sia/Skynet[18] and Filecoin[19] come closest to the idea of *Naeon Pool*, but neither offers all properties as summarized above.

IV CASE STUDY

The following real-world scenario gives us an idea of cybercriminals' typical modus operandi, and of our vulnerabability to such attacks. A small Kentucky based company that fell victim to a ransomware attack[20], decided to not get the authorities involved and to pay a $150,000 ransom instead, in order to regain immediate control of its data. Here's what happened. One Saturday morning, administration employees quickly jumped into alarm mode after they started receiving a simple, yet disturbing email that seemed to point in the direction of a cyber attack. The owner of this small company, whose IT infrastructure consisted of eight PCs, was hastily notified. The idea that his small business had come under the crosshairs of cyber criminals, baffled him. Until that moment, he had always thought such a thing impossible. All PCs were locked. The only thing they showed was an on-screen message: a ransom note with a phone number. The company's IT con-

tact quickly concluded that this was a real cyber intrusion, and advised to proceed with caution here. Although the ransom note showed a contact phone number, the IT contact advised to consult their insurance company and discuss a strategy with them before anything else. As this was the first time they faced such a situation, they thought it was of the utmost importance that they got it right first time: in many cases, things go astray when more than one interaction with the hackers is needed before reaching a successful settlement with them. The following two days were spent on damage assessment. One thing that raised eyebrows was the fact that the company was not holding any data that could be harmful to anyone when breached, like sensitive business information or privacy related data. The conclusion was that the hackers were only interested in money, and effectively shutting down the company by kidnapping its data was their only weapon. By then, the realization had set in that their business was totally dependent on its data, and without it, everything comes to a screeching halt. With 25 employees depending on this company to sustain their households, this is an absolute nightmare for every employer. It was now no longer a matter of *if*, but *when* to pay the hackers. Not paying the ransom would mean that the data would remain locked, so the insurance company started negotiations with the hackers. Initially, they demanded $400,000. Normally this gang asked for much higher ransom amounts (in the $1 million to $10 million range) but since this was a reasonably small company, the ransom amount was adjusted accordingly. Eventually, the hackers agreed on $150,000 in Bitcoins to unlock the data, a payment was made and the files decrypted.

"How could this have been prevented?" was the next burning question. The Kentucky company was told that for a business of their size, their systems were good, a few missed upgrades aside. Most likely, the hackers could simply find their way in "after someone in your office just clicked a link in an email."

V Conclusion

We have come a long way from the euphoric early days of the internet, when our digital curiosity did not stretch further than looking at a coffee pot at *Cambridge University's Computer Laboratory*[21], to our present day, increasingly hostile digital world where our data, our information, money, and even our identities get stolen when we leave the proverbial window ajar for a minute. Even though we know — or should know — perfectly well what to do in order to protect ourselves from cybercriminals, in the end we are all human beings with our human flaws. And sooner or later, hackers will find their way in through the tiniest hole in our defences, like brigands in the bushes laying siege, waiting for the first human flaw they can capitalize on. The best way to protect ourselves from digital disasters, apart from being vigilant and take all precautionary measures we can to prevent hackers to exploit our human flaws, is to encrypt and save our data where it is impossible for them to access it. Between vulnerability arises and disaster strikes is our last chance to prove that we are smarter than cybercriminals, by using their own weapon: cryptography. *Naeon* aims to be the tool that allows us to keep our data safe, even when we left the window ajar.

Appendix

This paper, as well as the latest version of *Naeon,* can be downloaded for free from *Naeon*'s website www.naeon.nl. *Naeon* has a SourceForge page at https://sourceforge.net/projects/naeon/.

Acknowledgment

The author would like to thank Dr Tim van der Avoird for his most valuable input during all stages of this work, Dr James Shuttleworth, Principal Lecturer at *Coventry University and Institute of Coding*, Dr Derrick Newton, Lecturer in Computer Science at *Coventry University*, and

Jonathan S. Weissman, senior lecturer in the Department of Computing Security at *Rochester Institute of Technology*, for their inspiring lectures that stimulated his intellectual curiosity and creativity in the fields of cyber security and cryptography. Also a word of thanks to Dr Eefje Vonken, Fatin Kumbasar and Angelica Andersson for their helpful comments and suggestions in the early conceptualization phase, and Peter Kras for his thoughts on the user interface.

REFERENCES

1 David Reinsel, John Gantz, John Rydning, *Data Age 2025: The Evolution of Data to Life-Critical*, IDC, 2017, p. 5.

2 446EB = 446 x 1000TB × 1000TB = 446 million TB (Terabyte).

3 Lily Hay Newman, *The Worst Hacks of 2020, a Surreal Pandemic Year*, Wired, 2020.

4 Courtney Heinbach, *New research: Datto's 2020 Global State of the Channel Ransomware Report*, Datto, 2020.

5 Given proper key management.

6 The number of permutations of *n* distinct objects is *n*! (the factorial of *n)*. The number of permutations grows exponentionally with *n.* For example: 60! is already larger than the number of atoms in the observable universe. The number of chunks that our encrypted data is split into, is a random number between 100 and 1000, which results into a number of permutations between 100! and 1000!.

7 The GNU Affero General Public License, the Free Software Foundation, https://www.gnu.org/licenses/#AGPL

8 (ISO/IEC 27000:2018, p. 2)

9 (ISO/IEC 27000:2018, p. 5)

10 (ISO/IEC 27000:2018, p. 2)

11 A 256-bit encryption key will have 115,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,640,564,-039,457,584,007,913,129,639,936 (that's 78 digits) possible combinations. No known supercomputer can crack that amount of combinations in any reasonable timeframe.

12 Let's imagine we have a truly random hash function that hashes from strings to n-bit numbers. This means that there are 2n possible hash codes, and each string's hash code is chosen uniformly at random from all of those possibilities. The birthday paradox specifically says that once you've seen roughly √(2k) items, there's a 50% chance of a collision, where k is the number of distinct possible outputs. In the case where the hash function hashes to an n-bit output, this means that you'll need roughly 2n/2 hashes before you get a collision. This is why we typically pick hashes that output 256 bits; it means that we'd need a staggering 2128 ≈1038 items hashed before there's a "reasonable" chance of a collision. With a 512-bit hash, you'd need about 2256 to get a 50% chance of a collision, and 2256 is approximately the number of protons in the known universe. The exact formula for the probability of getting a collision with an n-bit hash function and k strings hashed is: 1 - 2n! / (2kn (2n – k)!)
Source: https://stackoverflow.com/questions/62664761/probability-of-hash-collision

13 Assuming that the Master Key XML has not been breached.

14 It is thinkable that person A makes a backup with the single purpose of person B restoring it at some moment in the (distant) future. This way, Naeon could also serve as a communication method offering interesting application possibilities.

15 See Master Key XML, tag <Backup_timestamp>

16 See Master Key XML, tag <Selfdestruct_timestamp>

17 See Master Key XML, tag <Delete_on_retrieval>

18 David Vorick, Luke Champine, *Sia: Simple Decentralized Storage*, 2014.

19 Protocol Labs, *Filecoin: A Decentralized Storage Network*, 2017.

20 Jordan Hetrick, *Ransomware attack: Small Business With 8 Computers Paid Ransom*, PK Tech

21 Jim Boulton, *The first Web celebrity was a coffee pot*, Digital Archaeology, 2014

AUTHOR

René Smaal, rsmaal@naeon.nl