

Received August 15, 2019, accepted September 1, 2019, date of publication October 11, 2019, date of current version March 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2946730

# A Secure Data Infrastructure for Personal Manufacturing Based on a Novel Key-Less, Byte-Less Encryption Method

ANTON VEDESHIN<sup>1</sup>, JOHN MEHMET ULGAR DOGRU<sup>1</sup>, INNAR LIIV<sup>2,3</sup>,  
SADOK BEN YAHIA<sup>2,4</sup>, AND DIRK DRAHEIM<sup>5</sup>

<sup>1</sup>3D Control Systems, Inc., San Francisco, 94129 CA, USA

<sup>2</sup>Department of Software Science, Tallinn University of Technology, 12618 Tallinn, Estonia

<sup>3</sup>Centre for Technology and Global Affairs, University of Oxford, Oxford OX1 3UQ, U.K.

<sup>4</sup>Faculty of Sciences of Tunis, University of Tunis El Manar, Tunis 2092, Tunisia

<sup>5</sup>Information Systems Group, Tallinn University of Technology, 12618 Tallinn, Estonia

Corresponding author: Anton Vedeshin (anton@3dprinter.com)

This work was supported in part by the Astra6-1 project under Project 2014-2020.4/01.16-0032.

**ABSTRACT** We are witnessing the advent of personal manufacturing, where home users and small and medium enterprises manufacture products locally, at the point and time of need. The impressively fast adoption of these technologies indicates this approach to manufacturing can become a key enabler of the real-time economy of the future. In this paper, we contribute a secure and dependable infrastructure and architecture for that new paradigm. Our solution leverages physical limitations of the computational process into a defense strategy that makes distributed file storage and transfer highly secure. The main idea is to replace asymmetric or public-key encryption functions with an unkeyed, collision, second preimage, and preimage resistant cryptographic hash function. Such a cryptosystem does not have an inverse function  $H^{-1}$ . We challenge each block hash against the full hash table to recreate the original message. To illustrate the approach, we describe secured protocols that provide a number of desirable properties during both data storage and streaming. Similar to proof-of-work blockchain consensus algorithms, we parameterized the solution based on the amount of infrastructure available. Experiments show the proposed method can recalculate hashes for a 3-dimensional *live matrix* of  $256^3$  at an average of 14 revisions per second, and one revision every 5 minutes for a bigger matrix of  $4096^3$ . The increase in cloud infrastructure cost is insignificant compared to the level of protection offered.

**INDEX TERMS** Communication system security, computer aided manufacturing, content distribution networks, data security, data storage systems, distributed computing, information security, intelligent manufacturing systems, technology social factors, virtual manufacturing.


## I. INTRODUCTION

We are witnessing the advent of personal manufacturing, where home users, small and medium enterprises use devices such as 3D printers, CNC mills, laser jets, and robotics to manufacture products locally, at the point and time of need. The impressively fast adoption of these technologies strongly indicates that this novel approach to manufacturing can become a key enabler for the real-time economy of the future, i.e., a possible paradigm shift in manufacturing toward personal manufacturing. In such a paradigm, people

and organizations would not buy a ready-made product. Instead, they would obtain raw material and produce products using their own or locally accessible automated manufacturing (AM) machinery.

With the growing popularity of AM, robotic process automation (RPA), self-driving cars, automated medical devices, video and hologram streaming and internet of things (IoT) in general, the need to securely store and transfer streamable file types such as machine instructions and manufacturing files becomes more and more important.

Thus, the requirements for a modern secure distributed file storage and transfer are changing, and efficient methods of secured cloud storage and streaming are becoming a

The associate editor coordinating the review of this manuscript and approving it for publication was Raúl Lara-Cabrera .

compelling need. However, securing cloud file storage and transfer is a challenging task [1]. The nature and properties of modern files types impose certain constraints on how secure distributed file storage and transfer methods should operate.

One such constraint is the need to repeatedly access streamable files line by line or layer by layer without inconsistencies, delay, or compromising security through exposure of the whole file at once. In this paper, we address this problem and introduce a possible solution based on an efficient approach that utilizes technical limitations of the cloud and leverages them into a security control and defense strategy.

The main idea is to replace an asymmetric or public-key encryption functions with an unkeyed, collision, second preimage, and preimage resistant cryptographic hash function. Such a cryptosystem does not have an inverse function  $H^{-1}$ , and no key to decrypt the hash and get message back unless we pre-calculate a full hash table. We challenge each block hash against the full hash table to recreate an original message. To illustrate this approach, we have constructed secured protocols that provide a number of desirable properties to secure machine codes at rest and during delivery to stream consumption device.

The previous generation [2]–[4] of our solution has been implemented and proven over several years as a mechanism to securely deliver content to 3D printers from the cloud. Today, the 3DPrinterOS cloud has more than 84 000 users who have generated over three million CAD designs and machine codes. Users have produced more than 950 000 physical parts on 28 000 3D printers in 100 countries [2]; these values double every six months [2]. The technology is licensed to Bosch [3], Kodak [4], and other popular desktop 3D printer manufacturers. The solution described in this paper completely reworks the first [5] and second generation [2]–[4] of this secure content delivery mechanism and extends it to any type of manufacturing machine or complex IoT device with command, control, and telemetry.

The main contributions of this paper are: a) a novel, key-less, byte-less encryption method, ready for application to AM; b) an approach that leverages the physical limitations of the computational process [6] into a defense strategy; c) a threat model and security analysis of the proposed approach.

The main use case is the transfer of machine codes from secured cloud storage to a network-connected manufacturing machine. Other potential applications include streaming of a) video; b) holographic video; c) voice communication; d) medical data; e) business file data; f) telemetry, including command and control data to and from self-driving cars.

The remainder of this paper is organized as follows. In Section II, we introduce additional background and discuss the topics addressed in this paper. In Section III, we analyze and discuss why existing cloud file storage and transfer solutions such as digital rights management (DRM), video streaming and 3D model streaming fail to address critical constraints and security problems adequately. In Section IV, we explore a relatively new paradigm of cloud security, *live matrix*, proactive and passive cloud nodes, and our

protocol. In Section V, we thoroughly describe the proposed cloud application infrastructure and architecture; in Section VI, we discuss strong and vulnerable points of such an approach. In Section VII, we describe the setup used to evaluate the proposed method by conducting experiments with a local cloud of machines. Finally, Section VIII concludes the paper by summarizing the results and indicating issues to be addressed in future work.

## II. SETTING THE SCENE

This section prepares the reader for the proposed solution, which is described starting in Section IV.

### A. STREAMING VERSUS CONVENTIONAL SECURE FILE STORAGE/TRANSFER

#### 1) ARGUMENT: IMPORTANCE OF MACHINE INSTRUCTIONS

Seventy years ago, in the so-called “paper age,” most products’ technical drawings were prepared on paper. Imagine an attacker obtained pictures of the paper sketches of an innovative product. In the best-case scenario, it took many years to find or even build production technology, train engineers, set up a factory and production lines to produce prototypes and then a real product. In the worst case, there is no way to build the product using copies of the sketches, as the “secret sauce” required to build that product is somewhere down the production line, inside the heads and hands of the engineers working at a specific factory. A good example is rocket fuel; even with all the sketches of rocket structure and shape, people still need to identify and prepare fuel.

About thirty years ago, we entered the digital age, with the use of computer-aided design (CAD), computer-aided engineering (CAE), computer-aided manufacturing (CAM), computer-aided process planning (CAPP), computer-aided quality assurance (CAQ), production planning and control (PPC), and enterprise resource planning (ERP) tools [7]. However, these tools were initially used primarily to create a virtualization of a product to make measurements, manage bill of materials (BOM), and provide simulations to facilitate quicker changes to a product’s structure and shape during prototype testing cycles. Much manual work was still required, including post-processing and manual surface finishing. People are accustomed to using very basic solutions, like digital rights management (DRM), to secure CAD/CAM/CAE designs.

In the past, if such a DRM-protected CAD/CAM/CAE design was compromised, the barriers discussed above would still slow the rate of the product’s production and distribution. Compared to the “paper age” example, with decades required to produce the product, in the digital age, it might take only six months to figure out the details, find production facilities, and produce a marketable product.

In the personal manufacturing age, CAD/CAM/CAE intended for AM already has the “secret sauce” baked in. In other words, the proprietary information required to produce the market-ready product is inside the file. If such a

design is compromised, the attacker can reach the market with a production-quality product in just a few days, if not hours. Designs intended for AM and 3D printing contain all of the information needed to manufacture a real production quality product according to exact specifications: make and model of the manufacturing device, direction of layers infill, tolerances, surface finish, materials, speeds, temperatures, durability and taking into account force distribution and dispensation. With recent advances in AM technology, it is possible to manufacture a real working part or a usable product from a CAD/CAM/CAE design in just a few hours.

## 2) ARGUMENT: AN AM MACHINE IS A THIN CLIENT

The amount of information contained within modern CAD/CAM/CAE files for AM creates a load on the whole supporting infrastructure and requires substantial computing power. There is no way to put a supercomputer into each AM machine.

Over time, there has been a trend in AM to move as much calculation to the cloud as possible due to the low cost of cloud computing power. Initially, slicing for 3D printers was performed on the workstation built into a 3D printer (e.g., [8], [9]). Then, slicing software moved to engineers' workstations [10]. Now, slicing has moved to the cloud [2], with machine code streamed to the AM machine.

The next important step is to stream stepper motor pulses from the cloud directly to the AM machine. Firmware is moving to the cloud. As with software and faster computing, this move improves hardware operation, with incredible increases in quality and speed. For instance, Okwudire *et al.* [11] sent a low-level stepper motor commands from a server to simplified firmware, which interpreted simple commands and proxied them to the stepper motor drivers. They measured an increase in printing quality and speed.

AM machines should have a thin client built in, not a workstation [8], [9]. This thin client will interpret commands and send back current status and metrics. If the AM does not achieve a certain temperature or speed, the cloud needs to know, to update its manufacturing execution system (MES) and users about the delay. This approach will reduce costs and eliminate the need for local software updates. Moreover, the increase in calculation complexity possible in the cloud enables faster, smoother operation of local AM machines.

To explain why, we must first outline the basic steps that every contemporary AM machine firmware performs: a) read machine code into memory; b) interpret machine code into movements between coordinates; c) plan path through coordinates; d) calculate accelerations and decelerations with lookaheads taking into account inertia and potential forces; e) project movements to the stepper motor axis; f) ensure the motors and toolhead follow the programmed trajectory.

It is difficult to achieve excellent manufacturing quality when performing such processing on microcontrollers. Most firmwares perform only minimal prediction of the toolhead path. As a result, movement of the toolhead creates excessive vibration and noise, and it sometimes hits the wall of the

machine. These phenomena cause drops in manufacturing quality with any increment in manufacturing speed, despite the machines' excellent and frequently over-engineered hardware. The problem hides in the microcontrollers, which spend most of their computing time calculating trajectory. The less computing time the microcontroller spends on planning, and the more on operating the hardware, the better the manufacturing speed and quality.

To move the toolhead one millimeter, a stepper motor must perform a certain number of steps. For example, a 0.9 degree per step stepper motor performs a whole revolution in 25 full steps [12]. Such a motor will produce torn movements and generate substantial vibration. Moreover, the movements will be slow because of the inability to accelerate and decelerate efficiently; if configured to operate at high speeds, the machine will skip steps, resulting in missing manufacturing tolerances and overall lower product quality. The same motor operated with so-called micro-stepping, set at 1/32 of a step, will move much more smoothly, but require 800 steps per revolution [12]. However, not every microcontroller can maintain this rate of feeding steps into the motor driver. For context, an ATmega 16 MHz microcontroller with Marlin firmware achieves fewer than 10 000 steps per second (10 kHz) [13].

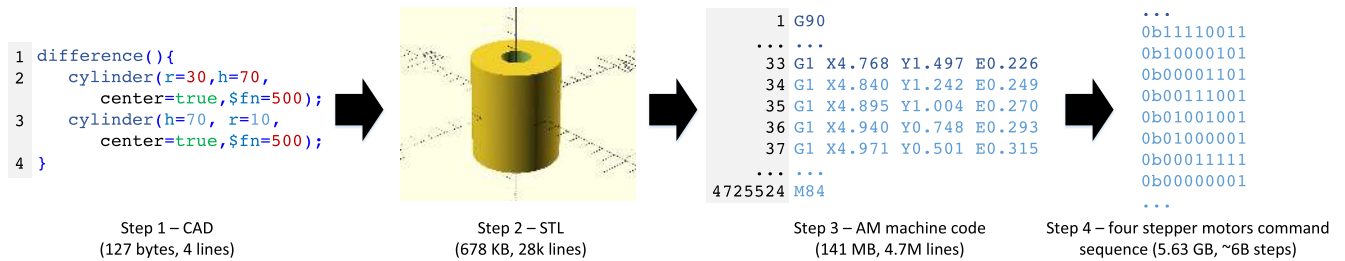
Moving path planning out of the firmware to a nearby computer increases manufacturing speed and quality. This was achieved by a team of researchers behind the Klipper project [14]. The same ATmega 16 MHz microcontroller described above, but operated with Klipper firmware [14], achieves 151 000 steps per second (151 kHz). It also drives the motors more smoothly, with fewer errors, and improved manufacturing quality. In the Step Benchmarks table [14] we can see that the same hardware can be 10x more efficient with the right software and more computing power. To achieve such improvements, we will ultimately stream encoded physical signal commands from the cloud to AM machines. The method proposed in this paper is ready for these types of applications.

## 3) ARGUMENT: LARGE FILE SIZES

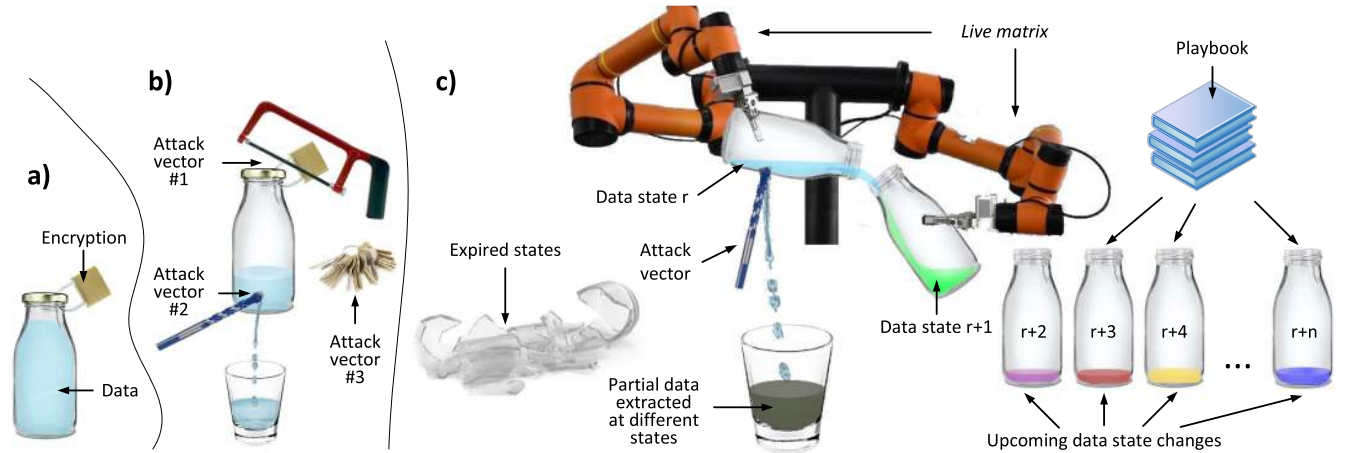
To explain why AM machine codes should be streamed versus downloaded and stored, we will use the example of a very simple 3D design—an annular cylinder—created in OpenSCAD software [15], [16].

The file for a given object will have a different size depending on which stage of manufacturing it is prepared for, and will involve different representations of the 3D object. We have depicted data file sizes at different stages of digital design for automated manufacturing in Fig. 1. As it shows, file size increases exponentially when moving from a less systematically specified representation of the object to the more specific representation needed to produce the production part.

In Step 1, the initial CAD design can be a few lines of code to mathematically represent a part. In Step 2, the STL file prepared for manufacturing is a set of triangles in space representing a CAD file; in addition to the overall shape of the



**FIGURE 1.** File size at different stages of digital design for automated manufacturing: From left: CAD design; an STL file prepared for manufacturing; machine codes for a specific AM machine make and model; command sequence for AM machine stepper motors. File size exponentially increases from the less systematically specified representation of the object to the more specific representation required to produce the physical part.



**FIGURE 2.** Representations of a) data encrypted with a static key; b) three attack vectors on static key encryption; c) dynamic key encryption with constantly changing data states—the state changes more quickly than the time required to physically extract the data.

object it contains information on manufacturing tolerances, the higher is precision the bigger is the file size. The lower the tolerances, the bigger the file size. In Step 3, machine code is produced from that file; this code is specific to a certain AM machine make and model. In addition to the shape of the object, it contains information about each individual layer the 3D printer will build to create the object. Each layer requires a certain number of movements of the toolhead. Each movement has an associated speed and information about the amount and speed of material extrusion. In Step 4, the command sequence for stepper motors file represents all of the signals that go to the stepper motor driver to execute the machine code. It includes calculations of acceleration and deceleration, takes into account inertia, timing, and many other factors. This is the exact recipe for how the part is produced. Changes in this last stage of preparation will affect the tolerances, quality, and speed of manufacture.

4) ARGUMENT: THE WHOLE FILE IS NOT NEEDED AT ONCE  
 In a past experiment [17], we found that a CAD file of a computed tomography (CT) scan of the human brain required about 2 GB, the corresponding medium-quality AM machine codes 6 GB, and print time for the full-size brain was 96 h. For a high-quality 3D print of the human brain, the machine code would be 36 GB, requiring approximately two weeks of manufacturing time on a 3D printer. The 3D printer did not need the entire file at once, as the manufacturing process

takes time, and it was possible to transfer the file in smaller segments.

### B. PHYSICAL LIMITATION OF COMPUTATIONAL PROCESSES

This is a basic example explaining how the physical limitation of computational process and different types of bottlenecks can be turned into a defense strategy in the cloud.

Let's use an analogy from the physical world. Let distilled water represent data we want to protect. A bottle of distilled water is put on a table, see Fig. 2 a). One approach to obtain the water without opening a lock on the bottle is to drill a small hole to let the water leak out (Fig. 2 b). Our storage solution could be compared with constantly changing bottles, and a robot which pours water from one bottle to another, adding and removing chemicals using different chains of chemical reactions to protect the water (Fig. 2 c). In this scenario, the water that is actually poured is, for example, sometimes a different acid, sometimes a different alkali. An attacker can still start drilling a hole in the bottle, but the bottle is still and steady only for a minute before the robotic arm starts to pour it to a different glass and add some other chemicals to change the state of the liquid. Only robot knows how many chemical transformations and in what sequence would lead back to the original distilled water.

If an attacker starts to drill holes into the bottles to steal the liquid, that attack requires time. If drilling a hole takes



5 minutes, and the bottle is only available in steady condition for 1 minute, then this is a clear bottleneck—a physical limitation. Now imagine a hacker used a faster way to drill a hole. It still takes time and there is a physical limitation—the diameter of the hole (in our approach, the network connection between the nodes and between the hacked node and secured cloud node). Now, the attacker starts to get a liquid. But if it takes, say, an hour to obtain all liquid from the bottle it will do the attacker no good—*this exact bottle holds the liquid for only 1 minute*—before the bottle is changed and the physical composition of the liquid is changed. The attacker has obtained some small amount of an unknown liquid, with no information about how to turn it back into its original form. By drilling subsequent holes and getting smaller amount of liquids at different stages of the chemical chain or recipe, the attacker will end up with a mysterious mixture with a complex chemical composition. The attacker will not know how to turn this mixture back into its original form. The attacker may have substantial time and computational power to analyze the liquid and to use brute force to get the original mixture. But this is near-impossible, as at some later time even the robot will not know what happened in the past; it does not have enough storage to keep versions of all of the obsolete recipes and chemical reactions. The faster the robot performs its manipulations, the harder it is to access the bottle for a reasonable amount of time, to drill holes or pump out the contents of the bottle.

Now, how does the solution described above translate to a computer problem? The metaphor described above with robotic arms and chemicals in bottles explains that it is hard to steal information that is constantly moving and transforming. This is the physical limitation. We compare our metaphoric example with what our solution does in Fig. 2:

- 1) A bottle with water and a lock on the lid **to** data store with data at rest, encrypted with a key (Fig. 2 a).
- 2) A drill bit, a key-ring with different master keys and lock picks, and a hacksaw attacking the locked bottle with water **to** encrypted data at rest and the use of various attack vectors to get to the data at rest (Fig. 2 b). This comparison represents an encrypted file in storage. Once an attacker gets a copy of the storage or the file, cracking it is only a matter of time.
- 3) Robotic arm **to** *live matrix* (Fig. 2 c). Our solution shuffles the data faster than an attacker can download it from the cloud, due to the physical limitations of computer systems, for example, the network interface.
- 4) Bottle with added chemicals **to** the data state in our solution (Fig. 2 c). The data state is static for a short amount of time, then it is changed. Within this short time period, it is hard to successfully extract the full file. The attacker ends up with partial data extracted at different states.
- 5) Broken bottles **to** expired data states (Fig. 2 c). If the data state is expired and not yet removed from the computer memory, it can no longer be used for retrieval of the data; thus, attacking it does not help crack the data store.

- 6) Drill bit **to** attack vector (Fig. 2 c). Any attack vector requires some amount of time to extract data. Before an attacker can extract the data, its state becomes obsolete, and the attack must be started from the beginning. Any attack through a computer system will face a physical limitation if the secured storage uses these physical limitations as a defense mechanism.
- 7) The bottle with the next chemical solution where the robotic arm pours the current chemical solution **to** current data state  $r$  and next data state  $r + 1$  (Fig. 2 c).
- 8) Queue of bottles with different chemical solutions according to a recipe **to** upcoming data states  $r + 2, r + 3, r + 4, \dots, r + n$  according to the *playbook* (Fig. 2 c).

### C. PUBLIC/PRIVATE KEY ENCRYPTION

Why not simply use public/private key encryption to protect manufacturing files? This approach is unfortunately prone to attacks in a manner similar to DRM. If a manufacturing file is encrypted with a static key, and the file is transferred and collected by the attacker, then decrypting it is only a matter of time.

One approach could use software like network security research tool Fiddler [18]. Fiddler can receive encrypted traffic using public-key, e.g., HTTPS traffic. When installed on a machine, it collects all dynamic public/private keys for all communication to/from that computer. It is relatively trivial to use an approach like this to collect dynamic keys and decrypt the files being transferred, without even compromising the software receiving the file. To compromise our solution, Fiddler would need to understand the in-memory *live matrix* data structure, understand how it is being calculated, and only then potentially perform an attack. This is a much more complicated scenario to execute compared to public-key encrypted file transferred over HTTPS.

### III. RELATED WORK

In this section, we present work that we consider to be close to the requirements described above and categorize relevant papers into six subcategories for a more systematic discussion. We start with some general considerations of cloud security, and then go more deeply into specific solutions, like point-to-point and point-to-multipoint secured communication, cloud secured storage, DRM, video streaming, and 3D streaming.

#### A. CLOUD SECURITY RISKS, REQUIREMENTS, AND MITIGATION

In [19], Brunette et al. provide a comprehensive analysis of possible issues in cloud security and how to mitigate them. They present a solid approach to assess existing cloud applications and provide a requirement base for the design of secure cloud solutions. That work provides notable recommendations. However, from our perspective, a next level—an integral solution—is necessary. For the sake of an ultimate security solution for cloud storage and file transfer,

we need a change in philosophy, and a new paradigm—*live matrix*—which we describe in Section IV-C.

### B. POINT-TO-POINT AND POINT-TO-MULTIPOINT SECURED COMMUNICATION

We examined related research on peer-to-peer, point-to-point and point-to-multipoint communication. First, most such solutions tend to use lower layers of the OSI model, mostly layer 3, the network layer. This positively affects the speed and throughput of the communication. At the same time it makes most of the protocols proprietary and exotic, which may make them hard to widely implement for AM machines. In contrast, the solution we propose in this paper is network layer and protocol agnostic, as the only information that is transferred is cryptographic hashes. Our solution would benefit from using a lower layer of OSI model, and streaming hashes over a lower level of the OSI model is a topic worthy of future research and experiments.

Second, the main efforts in the literature are focused on resolution of peers and finding and re-routing if a peer is disconnected. These mechanisms can compliment the solution described in this paper. Many approaches to point-to-point and point-to-multipoint communication security employ basic private/public key encryption, which does not prevent the exposure of intellectual property.

Mastorakis [20] and Mastorakis *et al.* [21] discuss peer-to-peer file sharing application designs and implementations that run on top of Named Data Networking (NDN). The security aspect is in the nature of the NDN architecture; however, this suggests cryptographically signing every packet in the network. NDN uses a distribution of data encryption keys as encrypted NDN data. Because it implements security at the protocol level, NDN offers good protection against negligence, in contrast to TCP/IP, where applications are responsible for security. Although NDN is considered to be the future of Internet [22], it is still at the stage of work in progress, and not yet ready for full production grade implementation.

### C. CLOUD SECURED FILE STORAGE AND STREAMING

In their cryptographic protocol [23], Jaatun et al. present an approach that is similar to ours. They segment files among the redundant array of independent net-storages in the computing cloud. The main thrust of their solution is the distribution of data across different cloud providers. Thus, the individual data deposits do not expose enough information about the owner and the file to make them vulnerable. In addition, in order to return the file to the user, the data must be reassembled. In our approach, we similarly distribute file parts to many machines in the cloud; however, we do not set a specific constraint on the form and number of cloud providers; our approach can utilize physical computing machines, virtual machines, Docker containers from one or several providers, etc.

Miller *et al.* [24] propose several robust security schemes for distributed file systems. They use segmentation of files

into file blocks, and file block encryption with asymmetric keys. Similar to [24], we split a file into segments and encrypt each segment with its own key. But we go beyond this, and propose a continuous re-encryption of file segments, with constantly changing keys. Moreover, we may constantly re-encrypt the symmetric keys that data segments are encrypted with. In our approach, re-encryption happens constantly on all cloud nodes at a preset file, computational, or cost limit.

In [25], Giuseppe et al. describe improved proxy re-encryption schemes for keys and apply them to secured distributed storage. We apply a similar approach in our solution, but to file segments, and not just keys. Furthermore, we re-encrypt continuously, regardless of reads and writes to storage. Cloud computing infrastructure prices drop each year; thus, such a re-encryption approach is feasible for use with millions of files.

### D. DIGITAL RIGHTS MANAGEMENT

There are many practical DRM-like approaches that are widely used in cloud storage and transfer. These include ECFS [26], and others mentioned in the same paper. In DRM, a file is usually encrypted using a symmetric or asymmetric key or a key combination before it is stored or transferred. In order to access the file, the data consumer needs the key. When an attacker obtains the key by, for example, buying the protected content once, brute force, social engineering, etc., then the file can be used or redistributed infinitely. DRM methods are usually lightweight and can be functional without any need for intensive cloud computing power. From our perspective, DRM methods are too vulnerable by their nature (Sec. II-C).

### E. VIDEO STREAMING

Numerous existing streaming approaches [27]–[31] work efficiently and consistently for video and music. Even though some of the protocols have consistency checks, they are not expected to deliver every single byte; insignificant data loss or delay caused by network problems is expected. However, this could be an issue for sensitive data, like CAD designs. For example, in the case of streaming designs to automatic manufacturing machines such as 3D printers or CNC mills, data transfer should be consistent and lossless: loss of a single byte while streaming is unacceptable, as this can lead to a AM machine malfunction or a defective product. At the same time, the streaming should be highly secured, which is not usually a requirement for media streaming protocols. In this paper, we show how to securely stream encrypted file segments directly from a highly secure distributed file storage.

### F. 3D MODEL STREAMING

In [32], Lin et al. describe a method to encode 3D models into a JPEG stream in order to transfer 3D designs. However, the solution is not comprehensive and has clear limitations.

In prior research [33], we theoretically described *live matrix* as a paradigm applied to secured 3D content delivery.

Our prior work is purely theoretical, and so lacks technical details and a real implementation of the method. This paper's contribution is to extend the initial idea with the necessary details for implementation and to technically broaden it to any type of secure file storage and transfer. Furthermore, we describe a threat model and conduct a thorough security analysis. It is worth mentioning that we eliminate the transcoding of files for streaming introduced in a prior work [5], [34].

In previous work [5], [34], we have explained in detail the necessity for secured streaming of 3D files and discussed methods to enforce 3D file copyrights. Our previous approach targeted a small niche case to secure 3D design transfer to 3D printers. That solution is very machine code-centric and lacks a tight coupling with the secured storage. Furthermore, it is vulnerable at the point of extracting a 3D design from the storage and re-encoding it for streaming. In the current paper, we propose a much more secure and consistent end-to-end method to store and stream files—regardless of file type—and without the need to re-encode the file for streaming.

#### IV. PROPOSED APPROACH

Relying on the principles and paradigms described below, we describe a working solution for highly secure distributed file storage and transfer.

##### A. ABSTRACT SOLUTION

For the cryptosystem [35]

$$D_d(E_e(m)) = m \quad (1)$$

where  $E$  is an encryption function,  $e$  is an encryption key,  $D$  is a decryption function,  $d$  is a decryption key, and  $m$  is a message, if  $d = e$ , then we have symmetric encryption. However, if  $d$  does not equal  $e$ , we have a public-key or asymmetric-key cryptosystem. The main feature of this cryptosystem is that only knowledge of the *static* decryption key is required to decrypt the message.

For unkeyed cryptographic hash function  $H$ , which is collision, second preimage, and preimage resistant [35]

$$H(m) = h \quad (2)$$

there is no such inverse function  $H^{-1}$ , and no key  $d, e$  to decrypt the hash  $h$  and get message  $m$  back:

$$H^{-1}(h) \in \emptyset. \quad (3)$$

In other words, a key for a hash does not exist.

Then, the only way to retrieve the original message is to hash all possible combinations and compare the hashes one by one. For example, if we know that the original message is five symbols from the ASCII table [36], given a strong cryptographic hash function [37] the only way to obtain the original message is to look the hash up in the table—the so-called brute force method [38]. To achieve this, we would need to create a hash table with  $_{256}P_5$ , a trillion elements, and then look up the original message by the hash. This

makes a brute force attack impractical, requiring substantial computational power.

Our solution is based on the complexity of retrieving the original message by its hash. To make the methods work, the task of our solution is to keep the complexity of the potential message set within a certain threshold, so just enough computing power is available to perform the calculations required.

The solution relies on a logic similar to that behind RSA SecurID tokens [39]. In that case, the same function with the same cryptographic seed is running on both the RSA server and the token (a small piece of hardware with a battery) in a user's pocket. In order to log in to the system, the user must enter a username, password and the code from the RSA token. The code on the RSA token expires every minute, and a new code is generated and is shown on the screen. A minute later, when the code expires, there is no way to reuse the code. In the proposed solution, we do something similar, but by recalculating a hash table and parts of the file on a regular basis—for example, every minute. After a minute, another hash table is calculated to accommodate file parts; the previous hash table expires and is deleted from memory. The process iterates over and over again.

In an abstract way, the solution works like this:

- 1)  $A$  is the (finite) set of symbols from ASCII table;
- 2)  $S$  is the (finite) set of file segments;
- 3) Each file segment  $s$  is set to a fixed length of  $m$  bytes;
- 4)  $t$  is a time variable and  $k$  is cryptographic salt.
- 5)  $G$  is the (finite) set of permutations of  $A$  set members with sample size  $m$ , so that  ${}_A P_m \in G$ .
- 6) *Sender and receiver side:* for each member  $g$  of a set  $G$ , together with time  $t$  and salt  $k$ , we calculate a corresponding hash  $h_{g,t,k}$  using hash function  $H$ . The hash is stored in a hash table  $T_t$  along with the original member  $g$ .

$$H(g, t, k) = h_{g,t,k} \rightarrow g \in T_t. \quad (4)$$

- 7) *Sender and receiver side:* when time  $t$  is incremented, table  $T_t$  expires at the moment  $t' = t + \Delta t$ ; Step 6 is repeated, and a new table  $T_{t+1}$  is calculated, so

$$T_{t'} \in \emptyset; T_{t'} \neq T_{t+1} \quad (5)$$

- 8) *Sender side:* for each member  $s$  of a set  $S$  we look up a corresponding hash  $h_{g,t,k}$  in table  $T$  using function  $L$  and send the hash to the receiving device. At this point we call hash  $h_{s,t,k}$  a *hint*. This *hint* does not contain any actual bytes from set  $S$ , and there is no key as such to decrypt the *hint* (see Eq. 3):

$$L(s, T_t) = h_{s,t,k}. \quad (6)$$

- 9) *Receiver side:* When the *hint* arrives, the receiver challenges it. The receiver performs a lookup against the local version of table  $T$  using function  $L$ . If such an element is found,  $L$  returns a file segment  $s$ ; otherwise, the value is undefined:

$$L(\text{hint}, T_t) = f; L(\text{hint}, T_{t'}) \in \emptyset. \quad (7)$$

10) The successfully received file is a set of *hints* positively challenged against the hash tables  $T_t, T_{t+1}, \dots, T_{t+n}$ .

In step 6, on the AM machine side, the same hash table with the same potential elements of set  $G$  should be generated in advance, taking into account exactly the same timing and salt (like RSA SecurID tokens have the same time-based function running on the server and the hardware token).

In step 9, when a *hint* arrives on the AM machine side, we look it up in the current hash table, and retrieve (or do not retrieve) the corresponding file segment. We recreate a file from successfully found segments. This is not a decryption function in terms of 1, as there is no key as such in terms of that equation, and actual static bytes are not transferred in its terms:

$$T_t \neq d; \text{Hint} \neq c \quad (8)$$

$$\Rightarrow L(\text{hint}, T_t) = f \neq D_d(c) = m \quad (9)$$

$$\Rightarrow L(L(s, T_t), T_t) = f \neq D_d(E_e(m)) = m \quad (10)$$

In the case of TLS/SSL, the actual encrypted bytes of the file are transferred. In our solution, only *hints*, which expire, are transferred. It is not possible to get a real byte of the file based on that *hint* a minute later. This is similar to the way in which an expired RSA SecurID code cannot be used.

In the next three sub-subsections we will explain important considerations about our approach.

### 1) SENDER AND RECEIVER SYNCHRONIZATION

Our approach is agnostic to synchronization method. Time  $t$  could be logical or physical time; in our experiments, we use physical time (UTC). Distributed machines can synchronize time against time servers. A minor change in time would not usually put the sender and receiver out of sync, unless the difference is larger than the *live matrix* state expiration time. For high latency networks or situations when time is slightly out of sync the *live matrix* expiration time could be increased so there is always a previous *live matrix* state available. At the same moment, there are two *live matrices* available—the current one and a previous one.

Dependency on time could be removed completely if we synchronize against other sources. Future accessible synchronization methods, might include natural phenomena like geomagnetic micropulsations [40], [41], seismic activity, gravity, blockchain block number, shared prior quantum entanglement [42], [43], and others.

### 2) COMPUTATIONAL AND BANDWIDTH OVERHEAD

Overall, cipher text has a positive difference in length between encrypted text and plain text.

Our solution has a bigger positive overhead compared to well-known stream ciphers [44]–[50], block ciphers [51]–[54] and plain text in terms of computation and bandwidth.

Commonly, in stream ciphers [55] the cipher text length has an insignificant positive overhead compared to plain text. A key is used to generate a stream, which is then combined

with the plain text to get the cipher text using an XOR operation. This does not significantly affect the amount of information transferred nor computation needed, as XOR is computationally inexpensive.

In block ciphers [56], padding is frequently added to plain text to make it equal to the block size, increasing the bandwidth overhead. Block ciphers also have computational overhead to encrypt each block compared to plain text.

Our solution has a parametric trade-off between computational complexity and security. By increasing the *live matrix* recalculation frequency, we increase the security level as well as the calculation complexity.

Our solution bandwidth overhead depends on the selected *live matrix* size and cryptographic hash function. The closer the number of bytes  $m$  to the output number of bytes of the hash function, the lower the bandwidth overhead. The recommended hash function output length should be close to the  $m$ , but not smaller than  $m$ . Overhead can be calculated as:

$$\text{overhead} = \text{length}(h_{s,t,k}) - \text{length}(m) \quad (11)$$

so that

$$\text{length}(h_{s,t,k}) \geq \text{length}(m). \quad (12)$$

The computational overhead of our approach is lower than that of block ciphers, and depending on the stream cipher algorithm, can be even smaller than stream ciphers. Hash function calculation is less expensive than AES and DES [57], [58]. Another possibility, which is highly application-dependent (e.g., not very practical for IoT and self-driving cars), is to scale hash calculation using a GPU and ASIC implementation of hash functions [59], [60]. However, block and stream encryption is difficult to implement using a GPU and ASIC-based approach.

### 3) CRYPTOGRAPHIC SALT

Salt  $k$  is not static. It changes with time, and could be an access code for a one-time manufacturing license, a PIN code, or part of a private key. Further, parameters other than  $k$  affect the setup; even if  $k$  is compromised, an attacker would still need to figure out the algorithmic setup and parameters.

## B. PHYSICAL LIMITATIONS OF THE COMPUTATIONAL PROCESS

Total security does not exist. Breaking into any system is just a matter of the time and money required to exploit its weaknesses. Indeed, cloud computing itself processes huge amounts of data in parallel, a capability that can be used against attacks. However, the storage, network and computing power of the cloud have physical limits to writing and readings files, transferring files over the network, calculating hashes, and encrypting or decrypting information. The philosophy behind our solution is to set an attacker versus a computing cloud and leverage the physical limitations of the computational process [6] as security controls. Similar to proof-of-work blockchain consensus algorithms [61],



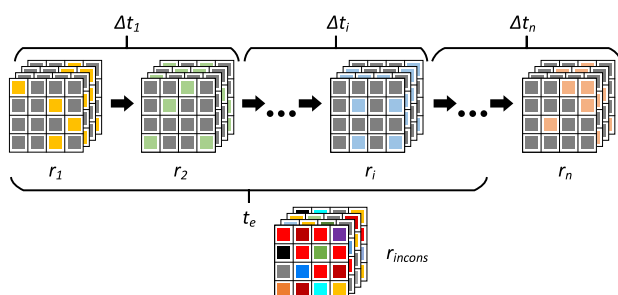
we parameterize the solution based on the amount of available infrastructure. The more computational power used, the harder and more expensive it becomes to carry out an attack.

Henceforth, we consider a hacker as a human individual, a group of hackers with special tools, or an automated script or bot with sufficient computing power. A hacker can never know all parameters and exact details of our secured cloud implementation, and it will take a considerable amount of time to find and exploit these weaknesses. This could be mitigated with detective cybersecurity strategies. If the hacker is equipped with comparable computing power, then the physical limitations of the computational process come into play.

There is physical latency at all levels of hardware and software during computational processes. In order to reduce latency, computer L1 and L2 cache memory is located very close to the processor [62]. The more distant some resource is from the processor, the higher the latency. For example, a network interface is usually a main bottleneck for distributed systems [63]. The operating system limit of open ports and I/O descriptors in Linux can be a bottleneck [64]. Our approach is to use these limitations and bottlenecks and turn them into a defense strategy.

**C. LIVE MATRIX**

*Live matrix* is a multidimensional data structure in which the data is constantly changing state. The state may even change millions of times per time frame,  $\Delta t_n$ , depending on the computing resources allocated. We refer to a different state during a certain time frame  $\Delta t_1, \Delta t_2, \dots, \Delta t_n$ , as to the revision  $r_1, r_2, \dots, r_n$ . The data in a *live matrix* are recalculated between revisions. The state of data in such a structure ideally changes more frequently and faster than the time it takes to extract the data from that structure. The period  $\Delta t_n$  during which *live matrix* is changing its state is much smaller than the period of time  $t_e$  needed to extract and store a single revision  $r_i$  of the matrix, as this would be a constantly moving target (Fig. 3). The data in the matrix are only consistent within one revision and become obsolete between revisions; thus, timing is crucial. The whole *live matrix* structure or any extracted file segments represent an inconsistent revision  $r_{incons}$  and quickly become obsolete.

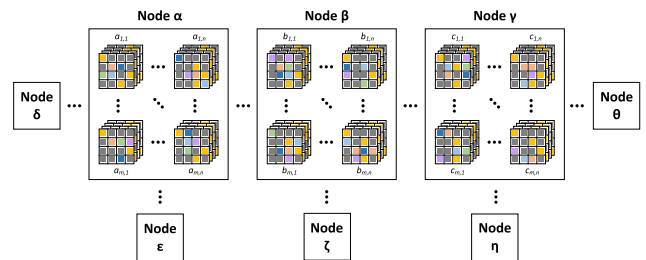


**FIGURE 3.** *Live matrix* state changes and inconsistent revision  $r_{incons}$ .

The matrix keeps multiple file segments, which reside in many locations of the matrix structure. These are encrypted and/or hashed using standard algorithms, e.g., AES256, 3DES, SHA-2, SHA-3, and located in the matrix at a certain index.

Taking into account the nature of the data to be hashed, i.e., the instructions for controlling the manufacturing machine, self-driving car, IoT or any other data, a special-purpose hash function can be designed. Matrix vectors that are not accommodated by useful data can be populated with fake data—random data that resemble the original file.

Distributed cloud storage can consist of one or multiple nodes  $\alpha, \beta, \gamma$ , etc. Each node may consist of one or multiple multidimensional matrices  $a_{1,1} \dots a_{m,n}, b_{1,1} \dots b_{m,n}, \dots$ , etc. (e.g., Fig. 4). The density of each matrix can be set from 0% to 100%. For example, if the density is 10%, then only this ratio of values are filled in with the real segments of files. The rest of the values are synthetically generated data or information very similar to the actual data.



**FIGURE 4.** Secured distributed cloud storage.

When the file is streamed from one location to another, the receiving location should also run a *live matrix* initialized with the matching encryption seed (based on time or other factors), and with a matching algorithmic setup. The stream comprises hashes of the file segment parts, *hints*. The actual information transferred in the stream is not the encrypted file parts, and there is no key to decrypt the streamed *hints* (unless not constantly changing its state matrix is considered a key). As soon as actual bytes of the file are no longer transferred, there is no key as such, and there is no function to decrypt *hints* (only to perform a look-up against the *live matrix* on the receiving end). We call this *key-less, byte-less information transfer* (Fig. 5).

When information is transferred to or from a non-cloud device (usually a data stream-consuming device with limited computing power, like a laptop, a manufacturing machine, a smart car, etc.) there is no physical possibility to keep more than a couple of revisions of live matrices on that side. It is thus impossible for such a device to decrypt the stream even thirty seconds later. This makes it impossible to “replay” the stream if an attacker records a fragment or even the whole stream.

**D. PROACTIVE AND PASSIVE CLOUD NODES**

A *proactive cloud node* is a cloud node or group of nodes that is autonomous to a certain extent. Proactive nodes do not expose any inbound TCP/UDP ports or APIs over a

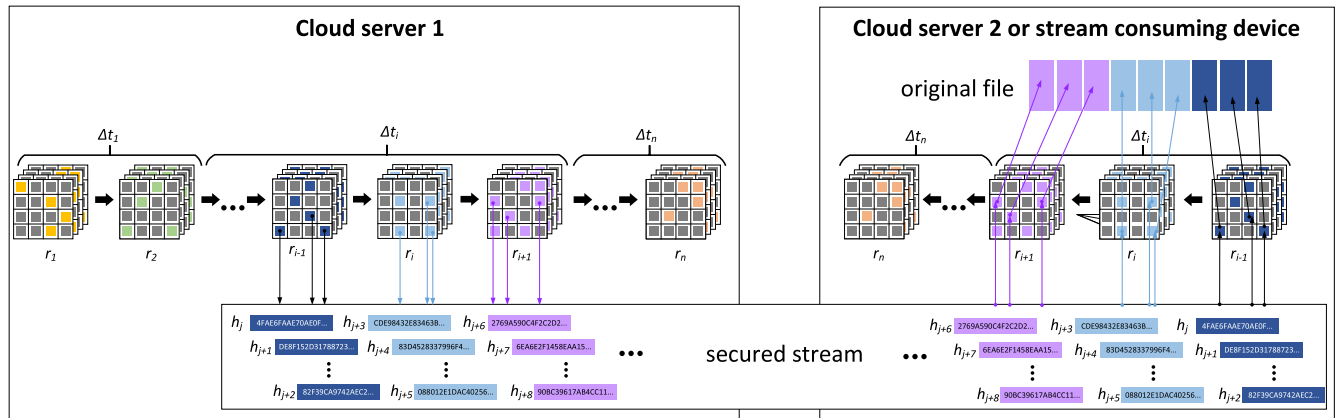


FIGURE 5. Key-less, byte-less secured streaming.

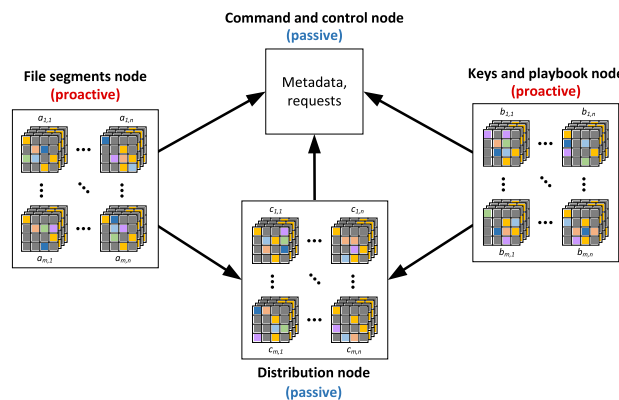


FIGURE 6. Types of nodes.

local or public network. Proactive nodes are the initiators of any communication between proactive and passive cloud nodes. Passive nodes cannot initiate the communication with proactive nodes; they need to wait for a request from one of the proactive nodes. Passive nodes only reply over the local network to requests incoming from proactive nodes.

Proactive nodes are used to store file segment data, users' public keys, file segments, encryption keys, and streaming playbooks. Passive nodes store metadata and run jobs, e.g., stream data outside the cloud, or deliver data at the right time at the right place, like a manufacturing machine or a self-driving car. The differentiation between proactive and passive cloud node types supports an important principle of segmentation in data security. Moreover, proactive cloud nodes rely on detective controls mechanisms [65] to analyze activity, events, logs, history of node communication, etc. Proactive cloud nodes can implement basic through the most sophisticated detective control methods using artificial intelligence, honeypotting, intrusion detection systems, etc. [66], [67].

E. PROTOCOL

A simplified protocol is presented in Fig. 7, 8, 9 and 10. All communication between cloud nodes is encrypted, although

this is not explicitly shown in the figures for the sake of clarity. Fig. 7 describes file upload by the user and secure storage of that file in the cloud. Fig. 8 and 9 describe storage maintenance over time and *live matrix* recalculation, respectively. There are two options to achieve the recalculation of storage: Fig. 8 depicts the use of a newly created set of keys, while Fig. 9 depicts utilization of the homomorphic properties of encryption methods. In the latter case, each file segment is recalculated by performing a homomorphic operation on a file segment. Thus, no additional key generation and exchange is necessary. The secured streaming protocol is depicted in Fig. 10.

V. IMPLEMENTATION

The highly secure distributed file storage and transfer solution setup (Fig. 6) consists of four types of nodes:

- a) The *command and control node* is responsible for storing command and control metadata. For example, whenever it is time to run a periodic job to re-encrypt file segments with a different set of keys, the file segments node and keys and playbook node communicate through this node
- b) The *file segments node* keeps the file segments in live matrices, performs a scheduled or on-demand recalculation of hashes or re-encryption of file segments, analyzes the behavior of the command and control node and distribution nodes, and makes corresponding decisions, for example, to support a streaming session initiated by the distribution node. Moreover, this node has controls that measure the speed of data consumption and compare it with realistic consumption rates. If the rate at which data are requested or consumed by the distribution node is faster than expected, an alarm state is triggered for a certain streaming session, or perhaps all sessions, depending on the setup and protection level desired
- c) The *keys and playbook node* is responsible for secure storage of keys and playbooks. Playbooks describe the sequence of segments in the file segments node. Without the right key, it is impossible to decrypt the file segment;

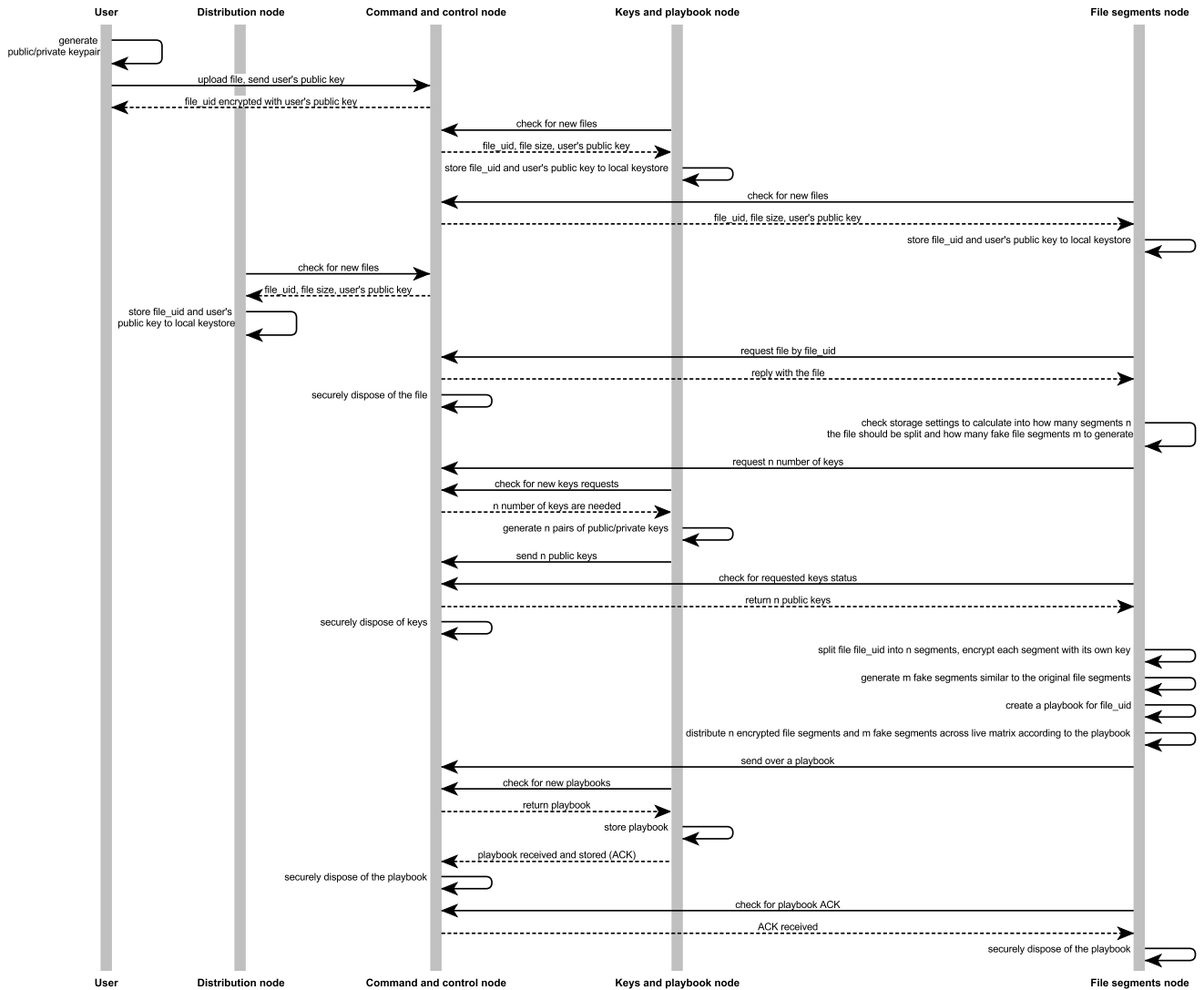


FIGURE 7. Protocol for storing a file in secured cloud storage.

conversely, without the right playbook, it is virtually impossible to locate and extract the desired data from storage. Depending on the setup, the keys and playbook node can deliver the right keys at the right time to the right place (e.g., to an AM machine which has already received a secured stream from the distribution node has recreated the file segments from *hints* using a locally running *live matrix*, and now needs to decrypt the data from file segments to produce the part). In the alarm state, this node stops the streaming process and stops issuing keys

- d) The *distribution node* runs content distribution jobs to transfer files to external sources, like other secured clouds or AM machines or self-driving cars. This node isolates different streaming jobs, optimizes streaming speed based on data transfer rate, and performs data delivery checks in the stream. It also participates in

the authorization scheme for external cloud and stream consumption devices.

The setup can be extended, so each node type is a sub-cloud of multiple machines implementing distributed live matrices (Fig. 4).

## VI. SECURITY EVALUATION

Our threat modeling and security analysis is based on several well-defined threat frameworks from Behl and Behl [68], [69], Behl [70], and Saripalli and Walters et al. [71]. The latter provides the list of “Threat events compromising cloud security” [71], which our distributed storage and transfer solution is intended to address. These are: a) *Isolation failure*: failure to effectively separate storage, memory, and routing causes isolation failure; b) *Malicious insider at cloud provider*: a cloud provider’s employee maliciously alters or corrupts customer data; c) *Intercepting data in transit*: failure

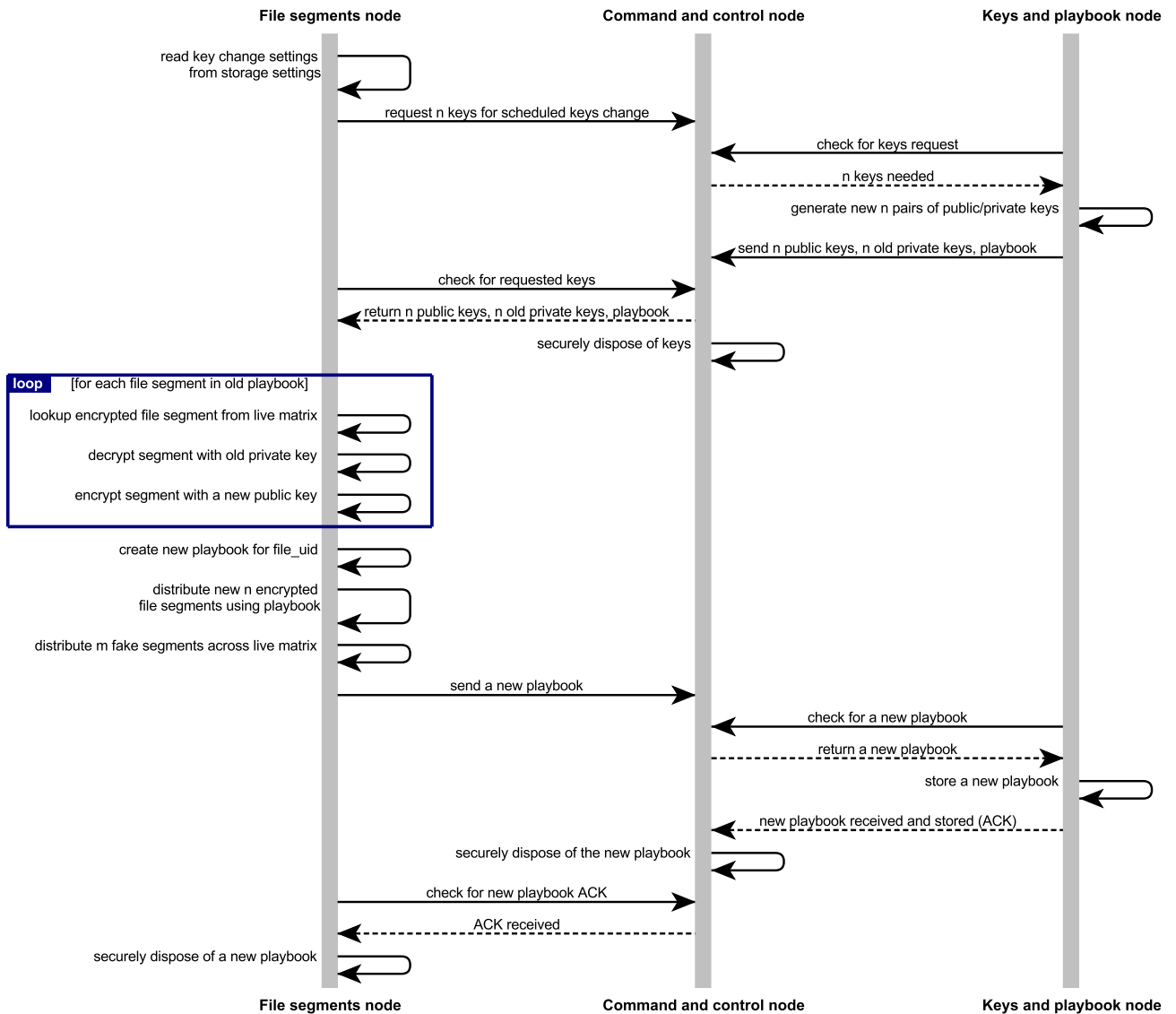


FIGURE 8. Protocol for storage maintenance and *live matrix* recalculation with key change.

in cryptographic techniques leads to data sniffing, spoofing and man-in-the-middle attacks during transit; d) *Data Leakage on Up/Down*: interception of data between the customer and the cloud provider leads to leakage of data to third parties; e) *Loss of encryption keys*: exposure of customer’s secret keys to malicious parties.

We have evaluated the relevant threats and created a threat model, summarized, along with the corresponding mitigation, in Table 1. We have derived the most important attack vectors from our threat model and provide an analysis.

If an attacker is able to pose as an authorized user, he still cannot download the data unless he digitally signs and submits a transaction to stream data to a data consumer.

There is no single point of compromise. If an attacker is able to access one of the node types, he still won’t be able

to extract data. An attacker needs to get access to at least two different types of node to decrypt the data. File segment nodes, keys nodes, and playbook nodes are proactive and do not expose any TCP ports. Thus, there is no way for the attacker to log in to these nodes unless they get access to a virtual machine or physical hardware and scan memory to get the contents from the running application.

If an attacker is able to obtain a playbook file, it is still only one instance. The attacker will not have access to every modified instance of the playbook. Without continuous updates, the attacker will not have access to the data.

Even if an attacker captures the data stream during a streaming session, it rapidly becomes obsolete very soon, unless the attacker obtained a seed to start and run *live matrix*. During a single streaming session, the data are from



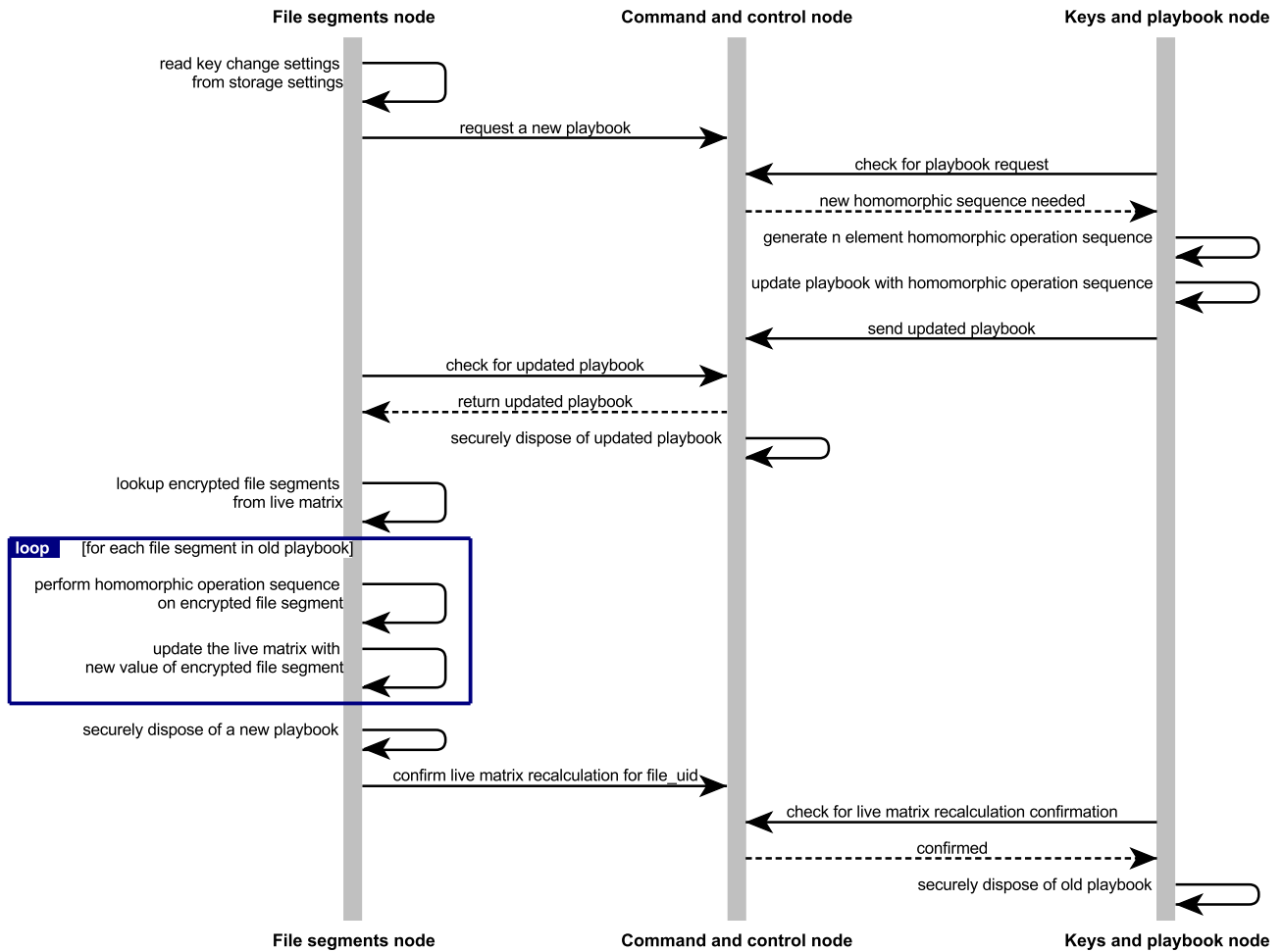


FIGURE 9. Protocol for storage maintenance and *live matrix* recalculation with homomorphic encryption.

different *live matrix* revisions, so in order to decrypt the data, corresponding *live matrix* states should be obtained. This can be done by compromising the server side or the data consumer side. This is still difficult; for instance, if the attacker gains access to the data consumer side, then he needs to be present from the very beginning of the stream and record the low-level machine code as it is transmitted. The solution depends on the exact data consumer implementation. For example, in the case of holographic video streaming, 3D printing, and other types of AM, data that are already consumed must be disposed of just after consumption. Additionally, if the attacker obtains one full unencrypted sequence of machine code, then this sequence could be used on exact make and model of the manufacturing machine, which makes it harder to distribute and violate the copyright.

In our previous research [5], [33], [34], we assumed that data—once taken from some kind of secured storage—are decrypted and then encrypted with a different method for delivery to the data consumer. Then, the distribution node can also be a point of attack. In that case, the attacker can obtain

a file or a stream on the server during re-encryption between storage and streaming. However, in the current approach, there is no need for transcoding the data.

In TLS file transfer, a certain key is used to encrypt data; if an attacker obtains the key, he can decrypt the file. Such keys are often reused by the services, or changed infrequently, making them vulnerable to collision and brute-forcing over time. In our solution, the complexity to brute-force the keys increases exponentially: the file is split into thousands of segments and the *live matrix* is constantly recalculated.

If an attacker obtains access to the data consumption device, he can receive file parts over a long period of time. There is no way to get all the files from the storage. Consequently, during one session, only one file can be obtained, and over a comparably long period of time. For example, producing a part using automated manufacturing can take days, and movies can last for hours. For an attacker acting this way, it would be inefficient time-wise to extract data from secured storage; this would not allow getting all the data from the secured cloud storage.

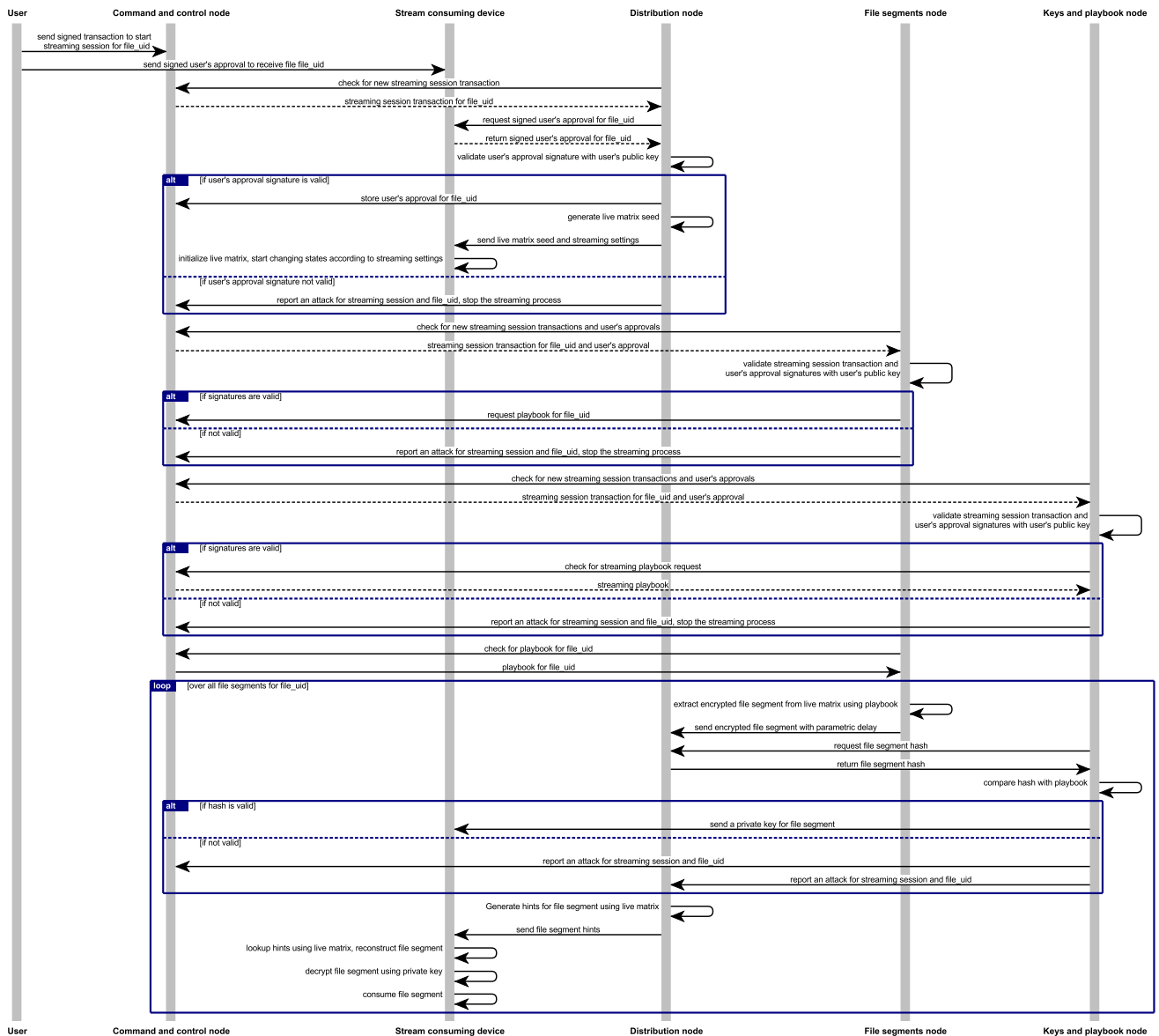


FIGURE 10. Protocol for secured streaming.

If an attacker starts to request more file parts within a shorter time-frame than a certain threshold, then the distribution node stops providing data. If such an attack is performed on a consuming device or a data channel, stopping the stream makes it impossible to get the rest of the file.

If an attacker carries out an attack on the secured cloud or a stream to a data consumer, secure cloud nodes collect the data and compare the metrics with those in configuration files. All abnormal activities, events, and logs can trigger an alarm state. A hacker would need to carry out a comprehensive analysis for a considerable period to figure out which changes in communication would cause an alarm state. By that time, the hacker would likely be detected, and mitigation procedures executed.

### VII. PERFORMANCE EVALUATION

In our lab-level implementation, we used the distributed database Cassandra [72] to implement live matrices, the Apache Spark near real-time distributed scale data processing [73] with the Java programming language to implement operations over matrices, and Apache Kafka [74] to maintain a queue of service requests and streaming jobs.

We stored file segments in column families of four bytes each, encrypted with public keys, in Cassandra. For public/private key generation, we used elliptic curve type secp256k1 [75]. We used the last 20 bytes of the public key to uniquely index encrypted file segments in the Cassandra column families. We used Apache Spark to re-encrypt the file segments and recalculate new indexes in Cassandra every two

TABLE 1. Threat model.

Threat	Mitigation
Cloud multi-tenancy	The end user does not have direct access to the data; he can only issue a start streaming command signed with his private key. There is no way to co-locate new malicious data with a victim's assets.
Elasticity	File segments are encrypted at rest and shuffled across the matrix; without a valid playbook, there is no way to extract the data. Moreover, after scheduled maintenance and recalculation of <i>live matrix</i> , the old playbook is discarded and a new playbook generated, so old data quickly become obsolete.
Availability of information	Although this particular method and its implementation does not currently provide comprehensive data backup options, it is intended for secured data distribution. The data owner should store a copy on offline media.
Cloud management layer	The solution is intended to run on cloud infrastructure such as AWS, Azure, or GCE, and this threat is mitigated by the cloud provider.
Information integrity and privacy	All transactions by the user to delete or transfer data from the secured cloud should be signed by a private key; then, nodes will independently verify the digital signature with the public key of the user. If the signature is not valid, no transfer is performed.
Cloud secure federation	The user uses a public-private key-pair to manage his resources at different locations. To perform streaming of the file to the data consumer, the user must sign a streaming transaction and send it to the secured cloud. A user can sign another transaction and send it to the data consumer, so the consumer can prove his eligibility to receive a secured data stream.

minutes, then provided the updated version of the playbook to the keys and playbook node. For hashing, we used the Keccak-256 [37] hash function.

We used a local cloud of four bare-metal physical machines to run the software. Two machines (one for the file segments node and one for the distribution node) each had an 8x GPU AMD Radeon RX580 chipset with 8 GB GDDR5, i7 CPU, 16 GB RAM, and 128 GB SSD. The other two machines, used for the command and control node and the keys and playbook node, respectively, had an Intel Celeron processor, 4 GB RAM, and 32 GB SSD. We set the GPUs in a computing mode and flashed them with a modified firmware for higher hash rates. On average, each GPU was able to produce 31.5 Mhash/s; a few outstanding GPUs performed at 28.5 Mhash/s. We achieved an average hash-rate of 248 Mhash/s total on each of the machines equipped with 8x GPUs.

First, we tested secured streaming between two cloud machines. We performed secured streaming of the 20 MB file in the local network from the file segment node to the distribution node. We were able to recalculate hashes for the three-dimensional *live matrix* of  $256^3$  at an average of 14 revisions per second. In another test, we were able to re-calculate a bigger matrix of  $4096^3$  with an average of one revision every 5 minutes. Second, we carried out the test between a cloud node and a data consumer node. For this test, we needed one more machine. The external stream receiving side was

TABLE 2. Proposed method performance (base rates 100 Mhash per h = \$0.05, 1 GB = \$0.087) for a 20 MB file size.

Experiment	Matrix size	State change frequency	$\Delta$ traffic (GB)	$\Delta$ cost (\$)
cloud-to-cloud	$256^3$	14 per s	1.67	0.1457
cloud-to-cloud	$4096^3$	1 per 5 min	0.85	0.0751
cloud-to-stream consuming device	$256^3$	1 per s	1.67	0.1437
cloud-to-stream consuming device	$4096^3$	1 per 64 min	0.85	0.0711

a laptop with an i7 processor, 8 GB RAM, and 256 GB SSD, GPU AMD Radeon RX570 chipset with 2 GB RAM, intended to emulate a single user consuming the stream. The GPU of this machine was able to produce 18 MHash/s. For this test, we needed to use only one GPU on the distribution node, with timing matching the calculation speed of the receiving machine. We were able to recalculate hashes for the three-dimensional *live matrix* of  $256^3$  states at an average rate of one per second. In another test, we performed a streaming session on the bigger *live matrix* of  $4096^3$ , and we were able to calculate a new state on average every 64 minutes. The results are reflected in Table 2.

We performed additional tests with different file sizes: 41 MB, 119 MB, 583 MB, and 1.1 GB. The results showed a linear dependency for overhead traffic and overhead server costs. In future research, we will seek to reduce overhead traffic.

The tests showed that overhead increases with smaller matrix sizes. This is a result of change in the matrix size to hash function output ratio in bytes. We recommend the use of proven SHA cryptographic functions, even if this creates a bigger overhead. If minimizing bandwidth is important, then hash functions with a smaller length output should be selected.

We also confirmed that the cloud can adapt to the computing power capacity available on the consumer's end and produce a stream that could be consumed with less computing power. With the increase in computing power needed to calculate *live matrix* revisions, the power needed by a hacker to try to decode the stream would increase exponentially. Our results show that catching such a stream would be an ever-moving target. Even in the case of success, the information would become obsolete very quickly, making it hard to carry out any analysis to decrypt the file being transferred.

## VIII. CONCLUSION

In this paper, we described and evaluated an approach that leverages the physical limitations of the computational process into a defense strategy to make cloud file storage and transfer highly secure. The method was designed to fulfill multiple important requirements for the use cases we discussed. The data transfer is lossless, so this method will work not only for delivering machine code to manufacturing machines, but for many other applications, including audio and video streaming. The most notable features of our

approach are: a) The solution is tightly coupled with secured storage, so there is no need for re-encryption in order to stream to remote data consumers, like other clouds or AM machines; b) By its nature, this solution keeps the data in partitions, and streaming also implies partition tolerance on file transfer. The data are segmented, and there are security controls based on the physical limitations of the computational process—it is not physically possible to extract and consume all the data within a reasonable time-frame; c) If multiple machines are used for each type of node, then there is no single point of failure in case of intrusion or fault; d) It may be used for peer-to-peer information transfer, though this requires the *live matrix* engine be installed on the peer machines participating in the information transfer; e) The solution can send bi-directional streams; on the receiving side, *live matrix* could be used for the reverse stream, for example, to transmit telemetry or interactive feedback.

In future work, we will concentrate on data storage fault tolerance mechanisms and intelligent adaptiveness to available computing power and network bandwidth.

## REFERENCES

- [1] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. IEEE Infocom*, Mar. 2010, pp. 1–9.
- [2] *3D Printers Cloud World Statistics*. Accessed: May 31, 2019. [Online]. Available: <https://cloud.3dprinters.com/dashboard/#world-statistics>
- [3] M. Molitch-Hou. (Jan. 2017). Dremel and 3D Printers Partner for 3D Printing in the Cloud. Engineering. com. [Online]. Available: <https://www.engineering.com/3DPrinting/3DPrintingArticles/ArticleID/14021/Dremel-and-3DPrinterOS-Partner-for-3D-Printing-in-the-Cloud.aspx>
- [4] S. Saunders. (May 2018). Thanks to New Partnership, 3D Printers Software Will now Power Kodak Portrait 3D Printers. 3DPrint.com. [Online]. Available: <https://3dprint.com/214798/3dprinters-kodak-portrait/>
- [5] K. Isbjörnssund and A. Vedeshin, "Secure streaming method in a numerically controlled manufacturing system, and a secure numerically controlled manufacturing system," U.S. Patent 2014 111 587 A3, Dec. 3, 2015.
- [6] R. Landauer, "Fundamental physical limitations of the computational process," *Ann. New York Acad. Sci.*, vol. 426, no. 1, pp. 161–170, 1984.
- [7] S. Kalpakjian, *Manufacturing Engineering and Technology*. London, U.K.: Pearson Education India, 2001.
- [8] *Slm280 2.0*. Accessed: Jul. 15, 2019. [Online]. Available: <https://www.slm-solutions.com/en/products/machines/slmr280-20/>
- [9] *Stratasys Objet1000 Plus*. Accessed: Jul. 15, 2019. [Online]. Available: <https://www.stratasys.com/3d-printers/objet1000-plus>
- [10] *Materialise Magics*. Accessed: Jul. 15, 2019. [Online]. Available: <https://www.materialise.com/en/software/magics>
- [11] C. E. Okwudire, S. Huggi, S. Supe, C. Huang, and B. Zeng, "Low-level control of 3D printers from the cloud: A step toward 3D printer control as a service," *Inventions*, vol. 3, no. 3, p. 56, Aug. 2018.
- [12] *Prusa Stepper Motor Calculator. Steps Per Millimeter - Belt Driven Systems*. Accessed: Jul. 16, 2019. [Online]. Available: <https://blog.prusaprinters.org/calculator/>
- [13] *Achievable Step Rates*. Accessed: Jul. 16, 2019. [Online]. Available: [https://reprap.org/wiki/Step\\_rates](https://reprap.org/wiki/Step_rates)
- [14] *Klipper 3D*. Accessed: Jul. 16, 2019. [Online]. Available: <https://www.klipper3d.org/Features.html>
- [15] Y. Nilsiam and J. M. Pearce, "Free and open source 3-D model customizer for Websites to democratize design with OpenSCAD," *Designs*, vol. 1, no. 1, p. 5, Jul. 2017.
- [16] M. Kintel and C. Wolf, "OpenSCAD," GNU General Public License, Tech. Rep., 2014. [Online]. Available: <http://files.openscad.org/mm.pdf>
- [17] A.-M. Nergi, (May 2013). *Eestis Pusti Pandud Innovatsiooniakadeemia Liigub Edasi Euroopasse*. [Online]. Available: <https://arielht.delfi.ee/news/uudised/eestis-pusti-pandud-innovatsiooniakadeemia-liigub-edasi-euroopasse?id=67036706>
- [18] *Telerik Fiddler*. Accessed: Jul. 17, 2019 [Online]. Available: <https://www.telerik.com/fiddler>
- [19] G. Brunette, *Security Guidance for Critical Areas of Focus in Cloud Computing v2.1*. Seattle, Washington, DC, USA: Cloud Security Alliance, 2009, pp. 1–76.
- [20] S. Mastorakis, "Peer-to-peer data sharing in named data networking," Ph.D. dissertation, Dept. Comput. Sci., UCLA, Los Angeles, CA, USA, 2019.
- [21] S. Mastorakis, A. Afanasyev, Y. Yu, and L. Zhang, "nTorrent: Peer-to-peer file sharing in named data networking," in *Proc. 26th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2017, pp. 1–10.
- [22] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. C. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.
- [23] M. G. Jaatun, G. Zhao, and S. Alapnes, "A cryptographic protocol for communication in a redundant array of independent net-storages," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2011, pp. 172–179.
- [24] E. Miller, D. Long, W. Freeman, and B. Reed, "Strong security for distributed file systems," in *Proc. IEEE Int. Perform., Comput., Commun. Conf.*, Apr. 2001, pp. 34–40.
- [25] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Trans. Inf. Syst. Secur.*, vol. 9, no. 1, pp. 1–30, 2006.
- [26] C. Rong and W.-C. Kim, "Effective storage security in incompletely trusted environment," in *Proc. 21st Int. Conf. Adv. Inf. Netw. Appl. Workshops (AINAW)*, May 2007, pp. 432–437.
- [27] V. Manasa and M. Vikram, "A secured adaptive mobile video streaming and efficient social video sharing in the clouds," *Int. J. Comput. Sci. Inf. Technol. (IJCSIT)*, vol. 5, no. 4, pp. 5153–5156, 2014.
- [28] M. Bucicoiu, M. Ghideu, and N. T. pus, "Secure cloud video streaming using tokens," in *Proc. RoEduNet Conf. 13th Ed., New. Educ. Res. Joint Event RENAM 8th Conf.*, Sep. 2014, pp. 1–6.
- [29] Z. Chen, H. Yin, C. Lin, and L. Ai, "3D-wavelet based secure and scalable media streaming in a centralcontrolled P2P framework," in *Proc. 21st Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, May 2007, pp. 708–715.
- [30] S.-H. Liu, H.-Y. Yu, J.-Y. Wu, J.-J. Chen, J.-L. Liu, and D.-H. Shiue, "A secured video streaming system," in *Proc. Int. Conf. Syst. Sci. Eng.*, Jul. 2010, pp. 625–630.
- [31] S. J. Wee and J. G. Apostolopoulos, "Secure scalable video streaming for wireless networks," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 4, May 2001, pp. 2049–2052.
- [32] N.-H. Lin, T.-H. Huang, and B.-Y. Chen, "3D model streaming based on JPEG 2000," *IEEE Trans. Consum. Electron.*, vol. 53, no. 1, pp. 182–190, Feb. 2007.
- [33] P.-M. Sepp, A. Vedeshin, and P. Dutt, "Intellectual property protection of 3D printing using secured streaming," in *The Future of Law and eTechnologies*. New York, NY, USA: Springer, 2016, pp. 81–109.
- [34] K. Isbjörnssund and A. Vedeshin, "Method and system for enforcing 3D restricted rights in a rapid manufacturing and prototyping environment," E.P. Patent 2 701 090 A1, Feb. 27 2014.
- [35] E. W. Tischhauser, "Mathematical aspects of symmetric-key cryptography," Ph.D. dissertation, Dept. Elect. Eng. (ESAT), Katholieke Universiteit Leuven, Leuven, Belgium, 2012.
- [36] V. G. Cerf, "ASCII format for network interchange," The Internet Engineering Task Force, Wilmington, DE, USA, Tech. Rep. RFC 20, 1969. [Online]. Available: <https://www.rfc-editor.org/info/rfc0020>
- [37] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak sponge function family main document," *Submission NIST (Round 2)*, vol. 3, no. 30, pp. 1–78, Apr. 2009.
- [38] B. Schneier, "Attack trees," *Dr. Dobb's J.*, vol. 24, no. 12, pp. 21–29, 1999.
- [39] A. Biryukov, J. Lano, and B. Preneel, "Cryptanalysis of the alleged securID hash function," in *Proc. Int. Workshop Sel. Areas Cryptogr.* New York, NY, USA: Springer, 2003, pp. 130–144.
- [40] J. A. Jacobs, *Geomagnetic Micropulsations*, vol. 1. Berlin, Germany: Springer, 2012.
- [41] R. A. Fowler, B. J. Kotick, and R. D. Elliott, "Polarization analysis of natural and artificially induced geomagnetic micropulsations," *J. Geophys. Res.*, vol. 72, no. 11, pp. 2871–2883, Jun. 1967.
- [42] R. Jozsa, D. S. Abrams, J. P. Dowling, and C. P. Williams, "Quantum clock synchronization based on shared prior entanglement," *Phys. Rev. Lett.*, vol. 85, no. 9, p. 2010, Aug. 2000.
- [43] M. Xu, D. A. Tieri, E. Fine, J. K. Thompson, and M. J. Holland, "Synchronization of two ensembles of atoms," *Phys. Rev. Lett.*, vol. 113, no. 15, Oct. 2014, Art. no. 154101.
- [44] C. De Cannière, "Trivium: A stream cipher construction inspired by block cipher design principles," in *Proc. Int. Conf. Inf. Secur.* New York, NY, USA: Springer, 2006, pp. 171–186.



- [45] P. Ekdahl and T. Johansson, "A new version of the stream cipher SNOW," in *Proc. Int. Workshop Sel. Areas Cryptogr.* New York, NY, USA: Springer, 2002, pp. 47–61.
- [46] J. D. Golić, "Cryptanalysis of alleged  $A_5$  stream cipher," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* New York, NY, USA: Springer, 1997, pp. 239–255.
- [47] M. Hell, T. Johansson, and W. Meier, "Grain: A stream cipher for constrained environments," *Int. J. Wireless Mobile Comput.*, vol. 2, no. 1, pp. 86–93, 2007.
- [48] R. Anderson and C. Manifavas, "Chameleon—A new kind of stream cipher," in *Proc. Int. Workshop Fast Softw. Encryption* Berlin, Germany: Springer, 1997, pp. 107–113.
- [49] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen, and O. Scavenius, "Rabbit: A new high-performance stream cipher," in *Proc. Int. Workshop Fast Softw. Encryption* Berlin, Germany: Springer, 2003, pp. 307–329.
- [50] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert, "Sosemanuk, a fast software-oriented stream cipher," in *New Stream Cipher Designs*. New York, NY, USA: Springer, 2008, pp. 98–118.
- [51] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2007, pp. 450–466.
- [52] J. Daemen, L. Knudsen, and V. Rijmen, "The block cipher square," in *Proc. Int. Workshop Fast Softw. Encryption* New York, NY, USA: Springer, 1997, pp. 149–165.
- [53] J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw, "The LED block cipher," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Cham, Switzerland: Springer, 2011, pp. 326–341.
- [54] B. Schneier, "Description of a new variable-length key, 64-bit block cipher (Blowfish)," in *Proc. Int. Workshop Fast Softw. Encryption*. New York, NY, USA: Springer, 1993, pp. 191–204.
- [55] R. A. Rueppel, "Stream ciphers," in *Analysis and Design of Stream Ciphers*. New York, NY, USA: Springer, 1986, pp. 5–16.
- [56] P. Mahajan and A. Sachdeva, "A study of encryption algorithms aes, des and rsa for security," *Global J. Comput. Sci. Technol.*, vol. 13, no. 15, pp. 14–22, Dec. 2013.
- [57] P. Trakadas, T. Zahariadis, H. C. Leligou, S. Vliotiotis, and K. Papadopoulos, "Analyzing energy and time overhead of security mechanisms in wireless sensor networks," in *Proc. 15th Int. Conf. Syst., Signals Image Process.*, Jun. 2008, pp. 137–140.
- [58] C. Xenakis, N. Laoutaris, L. Merakos, and I. Stavrakakis, "A generic characterization of the overheads imposed by IPsec and associated cryptographic algorithms," *Comput. Netw.*, vol. 50, no. 17, pp. 3225–3241, Dec. 2006.
- [59] P.-L. Cayrel, G. Hoffmann, and M. Schneider, "GPU implementation of the Keccak hash function family," in *Proc. Int. Conf. Inf. Secur. Assurance*. Cham, Switzerland: Springer, 2011, pp. 33–42.
- [60] L. Dadda, M. Macchetti, and J. Owen, "The design of a high speed ASIC unit for the hash function SHA-256 (384, 512)," in *Proc. Conf. Design, Autom. Test Eur.*, vol. 3, Feb. 2004, Art. no. 030070.
- [61] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 3–16.
- [62] J. R. Goodman, "Using cache memory to reduce processor-memory traffic," *ACM SIGARCH Comput. Archit. News*, vol. 11, no. 3, pp. 124–131, Jun. 1983.
- [63] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *J. Internet Services Appl.*, vol. 1, no. 1, pp. 7–18, May 2010.
- [64] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel: From I/O Ports to Process Management*. Newton, MA, USA: O'Reilly Media, 2005.
- [65] R. K. Ko, B. S. Lee, and S. Pearson, "Towards achieving accountability, auditability and trust in cloud computing," in *Proc. Int. Conf. Adv. Comput. Commun.* New York, NY, USA: Springer, 2011, pp. 432–444.
- [66] B. Nagpal, N. Singh, N. Chauhan, and P. Sharma, "CATCH: Comparison and analysis of tools covering honeypots," in *Proc. Int. Conf. Adv. Comput. Eng. Appl.*, Mar. 2015, pp. 783–786.
- [67] A. S. Sohal, R. Sandhu, S. K. Sood, and V. Chang, "A cybersecurity framework to identify malicious edge device in fog computing and cloud-of-things environments," *Comput. Secur.*, vol. 74, pp. 340–354, May 2018.
- [68] A. Behl and K. Behl, "An analysis of cloud computing security issues," in *Proc. World Congr. Inf. Commun. Technol.*, Nov. 2012, pp. 109–114.
- [69] A. Behl and K. Behl, "Security paradigms for cloud computing," in *Proc. 4th Int. Conf. Comput. Intell., Commun. Syst. Netw.*, Jul. 2012, pp. 200–205.
- [70] A. Behl, "Emerging security challenges in cloud computing: An insight to cloud security challenges and their mitigation," in *Proc. World Congr. Inf. Commun. Technol.*, Dec. 2011, pp. 217–222.
- [71] P. Saripalli and B. Walters, "QUIRC: A quantitative impact and risk assessment framework for cloud security," in *Proc. IEEE 3rd Int. Conf. Cloud Comput.*, Jul. 2010, pp. 280–288.
- [72] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.
- [73] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.
- [74] N. Garg, *Apache Kafka*. Birmingham, U.K.: Packt Publishing Ltd, 2013.
- [75] J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow, "Elliptic curve cryptography in practice," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* New York, NY, USA: Springer, 2014, pp. 157–175.



**ANTON VEDESHIN** received the M.Sc. degree in computer science from the Tallinn University of Technology (TalTech). He has developing software since the age of 11. He worked on Mitsubishi electrical car data mining project. He is currently a member of the National Doctoral School in Information and Communication Technologies. He is also a Lecturer of cloud computing course with the Tallinn University of Technology. He is also the CTO of 3DPrinterOS, working on industry applications with F500 enterprises. The technology stack includes programming languages, such as Java, GoLang, Python, PHP, and C++, Cloud technologies, such as Hadoop, Aerospike, Spark, Storm, Mahout, RapidMiner, Cassandra, Hbase, Solr, Vagrant, Docker: Mesos, Swarm, Kubernetes, Flocker, and Weave, blockchain, such as Ethereum, Hyperledger Fabric, and Corda R3, Ansible, Nagios, Redis, HAProxy, Nginx, and PaaS/IaaS, such as Amazon AWS, Microsoft Azure, and Google Cloud. His research interests include cloud computing, 3D printing, cyber security, blockchain, and data mining.



**JOHN MEHMET ULGAR DOGRU** is currently the co-founder and the CEO of 3DPrinterOS. He is an experienced start-up Founder and a former Lead Engineer at Dell, where he designed zero-time automated manufacturing systems. He has 20 years of experience in automated manufacturing and software development, IP security, and streamlining workflows.



**INNAR LIIV** was a Cyber Studies Visiting Research Fellow with the University of Oxford, from 2016 to 2017, a Visiting Scholar with Stanford University, in 2015, and a Postdoctoral Visiting Researcher with the Georgia Institute of Technology, in 2009. He is currently an Associate Professor of data science with the Tallinn University of Technology and a Research Associate with the Centre for Technology and Global Affairs, Oxford University. He also belongs to the Future

of Public e-Governance expert group at the Foresight Centre of the Parliament of Estonia. His research interests include e-government and data science, social network analysis, computational social science, information visualization, and big data technology transfer to industrial and governmental applications.



**SADOK BEN YAHIA** received the Habilitation to Lead Researches in Computer Sciences from the University of Montpellier, in April 2009.

He has been a Professor with the Tallinn University of Technology (TalTech), since January 2019, and the University of Tunis El Manar, since September 2001. His research interests mainly include combinatorial aspects in big data and their applications to different fields, e.g., data mining, combinatorial analytics (e.g., maximum clique

problem and minimal transversals), and smart cities (e.g., information aggregation & dissemination and traffic prediction). He has supervised 30 Ph.D. computer science students and more than 50 master's students. A selected list of his publications is shown at a glance through his DBLP website: [http://dblp.uni-trier.de/pers/hd/y/Yahia:Sadok\\_Ben](http://dblp.uni-trier.de/pers/hd/y/Yahia:Sadok_Ben).

Dr. Yahia is currently a member of the Steering Committee of the International Conference on Concept Lattices and their Applications (CLA) as well as the International French Spoken Conference on Knowledge Extractions and Management.



**DIRK DRAHEIM** received the Ph.D. degree from Freie Universität Berlin and the Habilitation degree from Universität Mannheim, Germany. He is currently a Full Professor of information systems and the Head of the Information Systems Group, Tallinn University of Technology, Estonia. The Information Systems Group conducts research in large and ultra-large-scale IT systems. He is also an Initiator and a Leader of numerous digital transformation initiatives.

• • •