Measuring Performance Overhead of Trans-encrypting HTTP Adaptive Streaming

A.T. Wiersma BSc

Master System and Network Engineering, University of Amsterdam, Netherlands
Medialab, TNO, Netherlands

6th August 2017

Abstract

The content distributors take content protection and Digital Rights Management of content very serious. As such many provisions already exist to make sure content is securely sent to, and only to, licensed clients. In this paper an alternative content encryption system is proposed by using the homomorphic features of asymmetric cryptography systems. Using a split-key derivative of the RSA cryptosystem, representing asymmetric split-key cryptography, the overhead of doing a trans-encryption operation on an edge was evaluated. The results show that using the split-key RSA cryptosystem degrades performance significantly over using a similar AES-128 re-encrypting cryptosystem but show area for improvement. Although the split-key RSA cryptosystem doesn't perform satisfactory it does provide the ability to distribute over untrusted third-party Content Delivery Networks (CDNs) as content is never decrypted along the pipeline, where this is the case for the similar AES-128 cryptosystem.

1 Introduction

Recent leaks of blockbuster titles have put pressure on streaming services and content distributors to better secure their stored and in transit video-content. The current Digital Rights Management (DRM) pipeline for HTTP Adaptive Streaming (HAS) works by encrypting video content once using Common Encryption (CENC) which relies on AES-128. This means the distribution of the content key or re-encryption is required when content is transfered to a third-party. Currently when content is distributed over a third party with a per client encryption means there will be a short time where content is available as clear text on the third-party machine. Alternatively the properties of trans-encryption's split-key cryptosystem could be used to forgo these issues as content would never be decrypted between the Content Provider (CP) and the Content Consumer Unit (CCU). This research aims to determine the overhead of doing a trans-encryption HAS video segments. This extra step of alternative encryption would allow distribution of DRM protected content over an untrusted third party. The development of trans-encryption over DRM is key to the CP because they rarely distribute their content themselves, while considerable value can be lost when leakage occurs at a Content Distributor (CD). As such an extension to a HAS protocol that enables the secure distribution of content over untrusted CDNs would greatly benefit the CPs. Trans-encryption would utilize the homomorphic properties of one of the following cipher-systems; RSA [1], One time path [2, pp. 398-400], LFSR stream cipher, ElGamal [3] or Damgard-Jurik [4]. To determine the viability of the technique a module doing the trans-encryption step was created and its performance overhead measured.

2 Research questions

As an important method for the content provider to protect the content during its distribution, trans-

encryption can provide a more secure environment for the delivery of digital content. This work aims to better understand the overhead of doing a trans-encryption step. As such the main research question for this project is proposed as follows:

– What is the performance overhead of doing a trans-encryption step for Dynamic Adaptive Streaming over HTTP (MPEG-DASH).

To fairly evaluate the performance of trans-encryption over MPEG-DASH the following subquestion corresponds to the main research question:

– How can currently available server hardware[1] be applied efficiently to do a trans-encryption step for MPEG-DASH.

In terms of scoping this research will focus primarily on evaluating the performance overhead introduced by the trans-encryption step on the HAS server.

This paper will aim to provide a clear measurement of the overhead of doing a trans-encryption step, and provide reference for the adoption of this technology.

3 Related Work

MPEG-DASH is the underlying HAS protocol that will be used as a representative of HAS technologies for this research. MPEG-DASH [5] was ratified as the ISO/IEC standard for delivering adaptive video content over HTTP in November 2011 [6]. Since then the DASH Industry Forum (DASH-IF) has further promoted the adoption of MPEG-DASH and continues developing the specification. The MPEG-DASH protocol does not specify a DRM method but supports all DRM techniques specified in the ISO/IEC standard for CENC [7]. CENC relies on a third party DRM license system to supply the Content Decryption Module (CDM) on the client's machine with keys to decrypt the content. The underlying encryption to these systems rely on CENC which encrypts content using AES-128 [8] Cipher Block Chaining (CBC) or Counter Mode (CTR). No underlying crypto algorithms other than CENC's AES-128 were found to have been researched.

In a HAS architecture the server has to provide a client with a number of different bit-rate videos to chose from. A mastervideo, which is preferably lossless, is encoded into a number of different bit-rates which correspond to a needed throughput by a client. These differently encoded videos are then described in a manifest which is usually text or XML. This process is described in Figure 1.

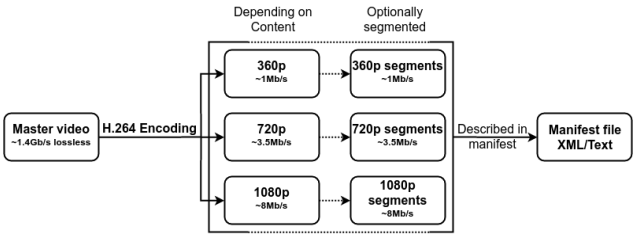

Figure 1: This diagram was created to show the abstract process of encoding video for HTTP Adaptive Streaming. There is an optional step in which the encoded video is segmented into equally timed parts. The timeindexes for the video are described in a manifest.

A client first requests the manifest from the server. Then a "slow-start", e.g. low bit-rate video first, is initialized to quickly fill up the buffer on the client. After this slow-start the video's bit-rate converges to the client's throughput and a client's optimal video quality is reached. When congestion takes place, the video bitrate is scaled back to decrease the possibility of playback discontinuity. This concept is shown in Figure 2.

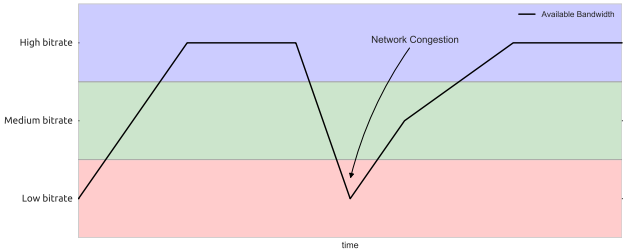

Figure 2: This diagram was created to show the abstract process of the adaptive algorithm switching bitrates following network congestion at the client-side for HTTP Adaptive Streaming.

Trans-encryption is a technique patented by TNO in WO application WO2013041394A1 [9]. It can be used to securely deliver content from a Content Provider to a Content Consumer Unit over one or more CDs.

[1] without adding dedicated processing units

These CDs are usually realized in the form of CDNs. Trans-encryption uses a split-key cryptosystem which splits up keys e and d for encryption and decryption algorithms E and D into i different split encryption keys $e_1, e_2, \ldots, e_i$ and or k different split decryption keys $d_1, d_2, \ldots, d_k$. The split-keys are generated such that $D_{d_k}(D_{\ldots}(D_{d_2}(D_{d_1}(E_{e_i}(E_{\ldots}(E_{e_2}(E_{e_1}(X))) = X$. The secret information S necessary for generating i and or k different encryption and or decryption keys has to be sent on together with the encrypted content. The point of this technique is that content X is never fully decrypted and remains encrypted on route to the CCU. An abstract schematic demonstrating trans-encryption for a single intermediary is shown in Figure 3.

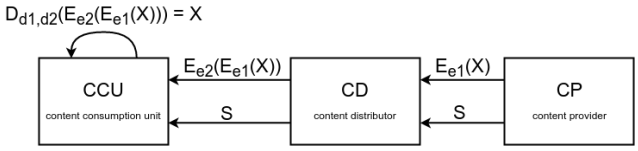

Figure 3: This diagram was created to show the encryption operations for a trans-encrypted stream of data to a client needed for a split-key cryptographic set-up.

4 Approach

As elaborated upon previously the MPEG-DASH specification and its stack will be used to represent HAS. MPEG-DASH utilizes a conventional HTTP server/client architecture to respectively distribute and ingest content. As such these two components needed to be selected and or created to simulate a representable set-up.

4.1 Client

To simulate load for the experiments the tool *wrk* [10] was chosen. *wrk* is able to do application layer requests in the form of HTTP requests with little to no overhead. By utilizing multiple concurrent connections *wrk* is able to simulate the load of multiple CCUs. *wrk* however was not used to decode the video segments as the requests were discarded on completion. *wrk* is able to evaluate the performance of a web-server in terms of throughput and latency which is returned to the calling program over the *stdout*. *wrk* also has an additional scripting interface in **LUA** [1] which allows the user to do intricate per request configuration. The measurement of the capacity of a web server is a tried and tested subject which generally focuses on either throughput or requests per second [11, 12]. *wrk* is perfectly capable of generating requests fast enough to sufficiently load web server. Because the focus of HAS web servers lies with providing sufficient throughput using large requests, throughput is the measurement that is focused on most.

4.2 Trans-encrypting Server

The second part to the HAS architecture is a HTTP server with exterior bindings to encrypting applications. These encrypting applications were specifically built for this research.

4.2.1 Encryption

As mentioned before there are a few possible cipher-systems available in the patent describing split-key cryptography. The patent suggests the use of the following five cipher-systems although the split-key cryptosystem is not necessarily constricted to this set.

- RSA [1]
- One time path
- LFSR stream cipher
- ElGamal [3]
- Damgard-Jurik [4]

RSA was chosen as the representative cipher-system for the experimental set-up's trans-encrypting cipher-system. RSA was chosen over the other cipher-systems because of its generally standardized status as a cryptosystem in multiple libraries. One time path was passed on because of its 100% increase in required data per trans-encryption and already has a base overhead of 100%. LFSR stream ciphers are generally fast but mostly because of dedicated hardware [13]. Many LFSR stream ciphers however are found lacking in security over the more secure asymmetric crypto algorithms [14, 15, 16]. Because this research focuses on currently available hardware for servers and because of the security aspect, LFSR stream ciphers were turned down. The ElGamal algorithm hangs on a different proof than RSA's practical difficulty of

[1] LUA is a lightweight programming language `https://www.lua.org/`

factoring, namely the indistinguishability of two probability distributions from cyclic group G [17]. Even though the ElGamal and RSA algorithm hang on a different proof, their performance is similar due to the asymmetric nature of the both of them. RSA is faster encrypting, ElGamal is faster decrypting [18, 19]. Because ElGamal is less standardized then RSA [20], RSA was eventually picked over ElGamal. Finally Damgard-Jurik, which hangs of the same principle as RSA, the difficulty of factoring, was not considered for this research because of the general lack of implementations as a crypto-system.

Split-key cryptography using the RSA algorithm applies the multiplicative property of consecutive exponentiations i.e. $(a^n)^m = a^{n*m}$ so that consecutive encryptions of X $(X^{e_1})^{e_2} \pmod n$ or consecutive decryptions of X $(X^{d_1})^{d_2} \pmod n$ can be combined into singular exponentiations of X: $X^{e_1*e_2} \pmod n$ & $X^{d_1*d_2} \pmod n$. To implement RSA trans-encryption using split-key cryptography first keypair one is generated following the normal key generation procedure described in the RSA standards. Then keypair two is derived copying keypair one's public modulus n, its prime p and q. For keypair two a different public exponent is chosen from which private exponent two is derived. These two public/private keypairs can then be combined by multiplication as follows: $e_c = e_1 e_2$ and stored in a combined keypair. Because there are two possible operations for the trans-encryption e.g. a second encryption or a first decryption, this leaves two possible operations on the trans-encrypting server this decision can be seen in Figure 4.

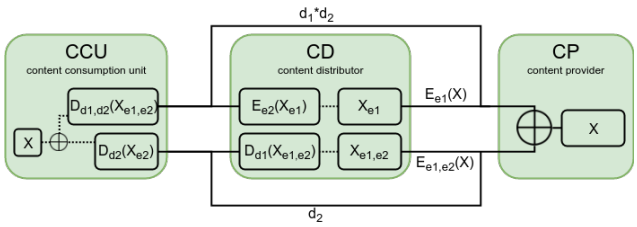

Figure 4: This diagram was created to show the encryption and decryption routes possible when doing a RSA trans-encrypting content pipeline.

For this research four components to the RSA trans-encryption cryptosystem were created. These components were created using the latest version of the **C** *openssl* library. Using the *openssl* means forward compatibility and efficient use of the hardware with the use of its RSA and AES bindings. The four components were built to work on generic content.

- **C** *rsa_create_combined*
- **Python** *encrypt.py* + **C** *rsa_encrypt*
- **C** *rsa_trans*/*rsa_trans_dec*
- **C** *rsa_client_decrypt*

After *openssl genrsa* is called to create a keypair one *rsa_create_combined* is used. *rsa_create_combined* creates a second keypair from the previously created pair one and combines it in a combined pair. These pairs are all stored in a **PEM**[1] formatted file. *encrypt.py* and *rsa_encrypt* are used to encrypt entire directories of video content with either the combined keypair or pair one applying PKCS1 padding to every consecutive 245 bytes of a HAS segment. The 245 bytes were chosen as PKCS1 padding allows for padding of $keysize - 11$ bytes of data to be encrypted per call of the RSA algorithm. *rsa_trans* or *rsa_trans_dec* are used respectively to either apply the second encryption using keypair two or apply a first decryption using keypair one. This step does not apply any padding. Finally *rsa_client_decrypt* performs the final decryption using either the private exponent from the combined pair or pair two. The two different paths can be seen as follows:

$$E_{e1} \rightarrow E_{e2} \rightarrow D_{d1d2}$$

$$E_{e1e2} \rightarrow D_{d1} \rightarrow D_{d2}$$

Before continuing the functionality was confirmed in Linux using a MPEG-DASH segment as follows:

1. ./rsa_encrypt 0001.m4s > 0001.one.m4s
2. ./rsa_trans 0001.one.m4s > 0001.two.m4s
3. ./rsa_client_decrypt 0001.two.m4s > 0001.clear.m4s
4. diff 0001.m4s 0001.clear.m4s (no output means no difference)

[1]PEM describes a way of the textually encoding cryptographic keys, certificates, and other data. PEM is the most used format for distributing ASN.1 DER keys and certificates.

The functionality of the passthrough part of the HAS server was confirmed beforehand by having an actual MPEG-DASH client[1]communicate with the server.

As a comparative technology an AES-128 re-encrypting application was also implemented using the **C** *openssl* libraries. This implementation first does a decryption with key one using AES-128-CBC, which is then followed by an encryption with key two. The subsequent decryption and encryption do possess a small time window in which the content is available on the re-encrypting machine as cleartext.

4.2.2 HTTP-server

The HTTP server receives the requests from *wrk* and on the fly does one of four operations on the MPEG-DASH segments:

1. A passthrough, as a baseline. (no-op)
2. A Re-encryption step, which involves a decryption and an encryption applying CENC.
3. A trans-encryption step, which does a second encryption using RSA.
4. A trans-encryption step, which does a first decryption using RSA.

The web-framework used as the (trans)encryption/passth middleman is *Japronto* [21], a low-overhead python HTTP-toolkit. *Japronto* is a HTTP web-framework that was coded in C which with extensive use of the Python/C bindings has been made easily programmable using a Python interface. The C-code for *Japronto* is tweaked to optimize for modern CPUs. *Japronto* uses *picohttpparser* [22] for header & chunked-encoding parsing and uses *uvloop* [23] to provide with asynchronous IO. When able writes are combined to save system calls. Japronto uses a master slave set-up to distribute incoming connections over available slaves.

4.3 Experimental Set-up

The experimental set-up consists of the two previously proposed components hosted on two separate machines connected by a 1Gbit link. A schematic overview of this set-up can be seen in Figure 5.

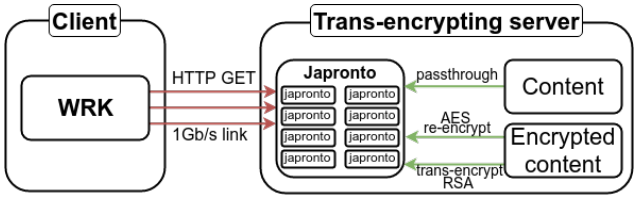

Figure 5: This figure was created to show the experimental set-up for the benchmarking measurements. This set-up allows testing of the multiple different cryptosystems.

The keysize for RSA was chosen as 2048-bit, this was done to ensure that factorization of the key would be impossible for the foreseeable future. Figure 6 shows the progress of hostile RSA factorizations since 1992, which indicate the possibility of factorizing RSA with a 1024 bit keysize by 2016. The choice for a 2048 bit keysize means that only the private-key for the public/private key pair is 2048 bits, but the public key can be chosen to be as small as possible to speed-up the encryption. For most RSA implementations the default public exponent e is chosen as 65537 [24], so this is what was done for this experiment as well. The main reason for not picking a smaller number like 3 is that it is unsafe when no padding is used.

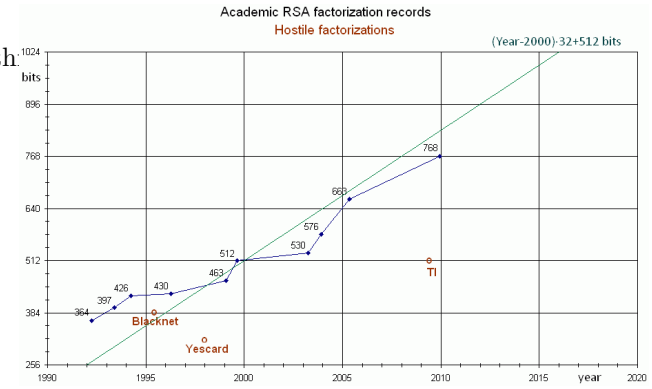

Figure 6: A figure showing academic factorizations of the RSA cryptosystem from 1992 till 2010 with a linear regression of the datapoints[1].

The relevant specifications for the two machines are as follows:

[1] `http://dashif.org/reference/players/javascript/v2.5.0/samples/dash-if-reference-player/index.html`
[1] Figure retrieved from `https://crypto.stackexchange.com/a/1982`

Client - Lenovo Thinkpad T440s

- Intel Core i5-4200U @ 2C4Tx1.6GHz
- Intel Ethernet Controller I218-V @ 1Gb/s

Trans-encrypting server - Dell PowerEdge R210

- Intel Xeon CPU E3-1240L v5 @ 4C8Tx2.10GHz
- Broadcom NetXtreme BCM5720 @ 1Gb/s

The number of threads on the server machine allows for 8 *Japronto* slave processes to be run concurrently with a one to one mapping on the CPU threads.
wrk is loaded with requests corresponding to MPEG-DASH segments over its LUA API. *wrk* is then run using 1, 10, 100 and 1000 concurrent connections for 120 seconds. These tests are run once for 120 seconds for each of the four possible operations the HTTP-server is capable of. Out of the results from *wrk* throughput to the server will be recorded and plotted.

The test content that was used is Tears of Steel, an open source movie by the Blender foundation. It was encoded using H.264 in the following bit-rates:

- 388227 bits/s with a resolution of 512x288 pixels.
- 770624 bits/s with a resolution of 640x360 pixels.
- 1148766 bits/s with a resolution of 852x480 pixels.
- 1996574 bits/s with a resolution of 1280x720 pixels.
- 2823478 bits/s with a resolution of 1920x1080 pixels.

The urls corresponding to the video segments were randomly shuffled once before testing began. Then the randomized list of urls were requested from each of the different encrypting set-ups.

5 Results

This section shows the results for the experiment described in Section 4.3. The results show the performance under a number of different concurrent connections, helping understand the overhead induced by a trans-encryption step.

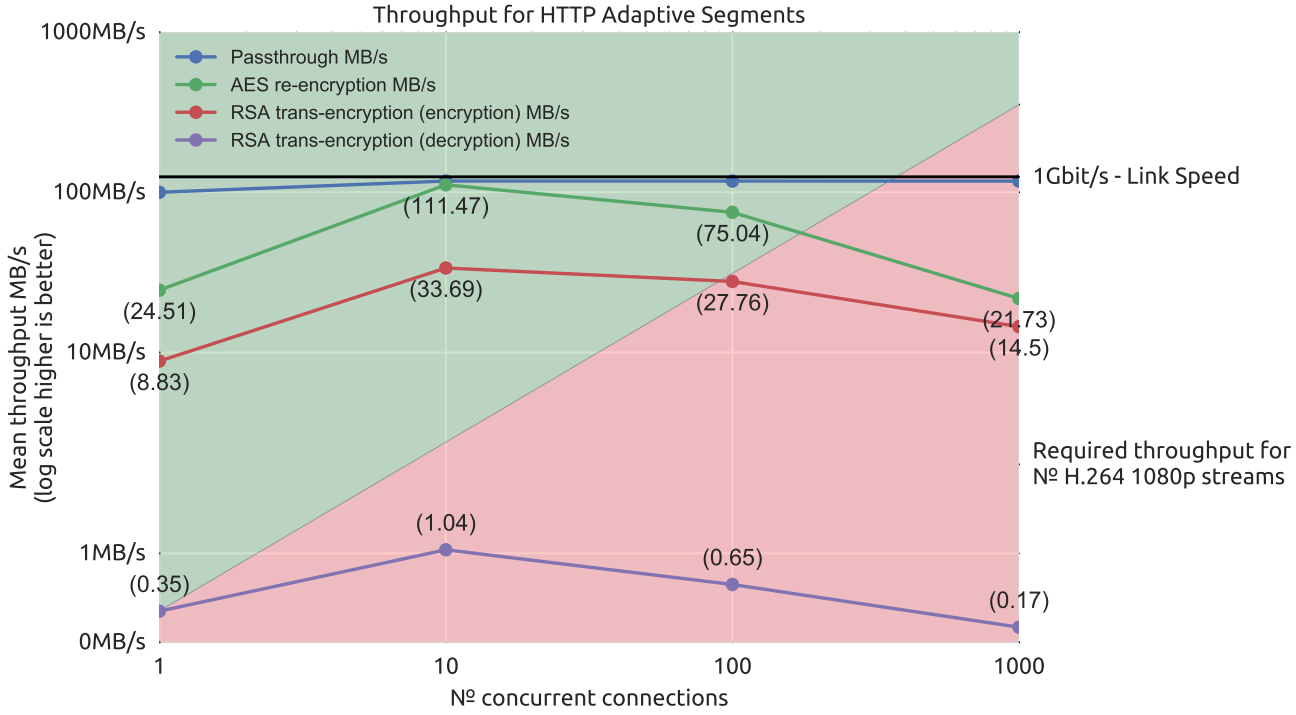

Figure 7: A graph showing the throughput of the experimental set-up for different types of encryption over MPEG-DASH segments.

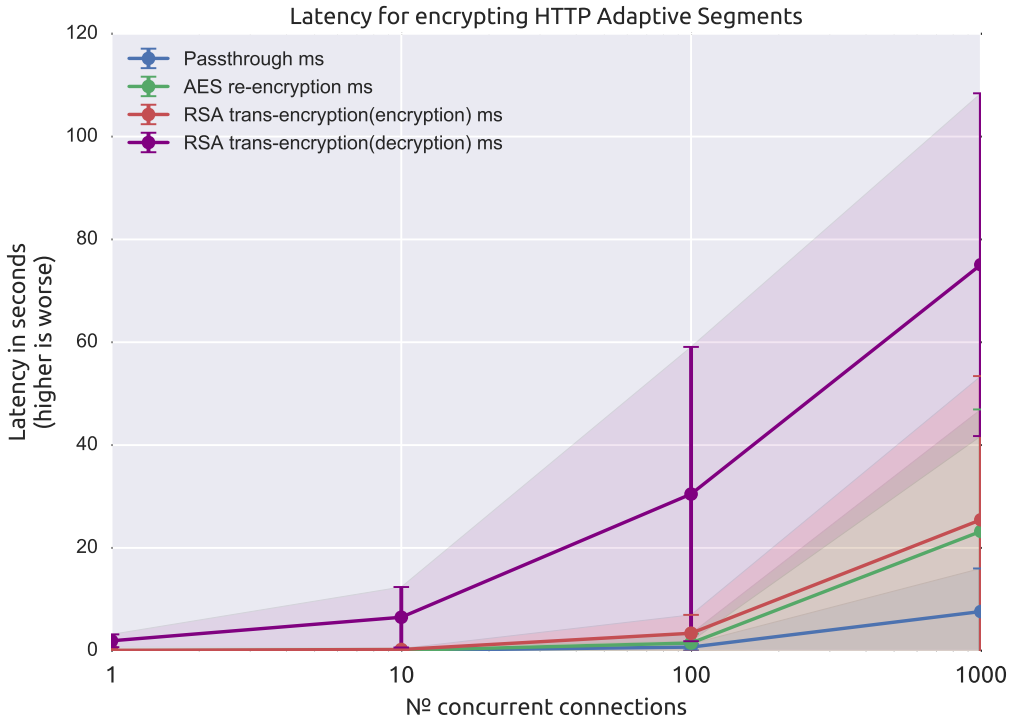

Figure 8: A graph showing the latency of segment transmission completions for different types of encryption over MPEG-DASH.

Figure 7 shows that from 10 connections and up *japronto* is generating passthrough responses at link speed which has been indicated in the graph by a black line. The passthrough experiment shows no slowing down over the increased connections which means the server is never overloaded in any way. The passthrough implementation should be capable of providing enough throughput for approximately 390 clients at 1080p H.264.

AES re-encryption shows the least amount of overhead in comparison to the passthrough baseline with a 51.3% average performance of the baseline. The amount of throughput provided by the AES re-encryption implementation should easily be enough to sustain 100 concurrent clients at 1080p H.264. RSA trans-encryption with an encryption step has the least amount of overhead of the two trans-encryption implementations with a 18.7% performance of the passthrough baseline. The amount of throughput provided by the RSA trans-encryption with an encryption step implementation nearly provides enough throughput to sustain 100 concurrent clients at 1080p H.264 and misses this mark by $\sim 3MB/s$. RSA trans-encryption with a decryption step has the most amount of overhead of all encrypting implementations with only 0.5% performance of the passthrough baseline. The decrypting implementation is only able to provide enough throughput for 1 client at 1080p H.264 after which scaling is not significant.

Figure 8 reestablishes the aforementioned overhead distribution, with the most latency in the trans-encryption implementation with a decryption step. The figure shows the average completion time for the requests send by the *wrk* client. The error bars signify the standard deviation of each of the averages plotted in the graph. In this figure you can also see that the latency of the passthrough implementation is actually increasing so requests are kept waiting longer as the number of concurrent connections is increasing.

6 Discussion

The AES re-encryption benefits greatly from the AES-NI set that the Intel Xeon E3-1240L CPU on the server provides. This extra skews the results in favour of AES re-encryption. It is not unthinkable that the instruction set could be extended to optimize streaming video performance for RSA decryption as hardware accelerated video decoding is a well known constant on modern CPUs.

6.1 Future Work

Future research could go into a RSA decrypting client to complete the alternative DRM pipeline and further evaluate the viability of the RSA trans-encryption. As an alternative to the CPU bound instructions utilized by openssl the RSA operations could be moved to a GPU. Previous research achieved a 2.17× speed up of concurrent RSA decryptions and a 1.572× speed up of concurrent RSA encryptions over a sequential CPU RSA implementation [25]. Because of the high number of compute cores in a GPU the RSA algorithm should scale better than a CPU at concurrent encryptions and decryptions. Although server machines do not usually carry dedicated GPUs, this could be facilitated to achieve a trans-encryption speed-up.

7 Conclusion

This research has evaluated the overhead of trans-encrypting RSA as an alternative encryption for the DRM pipeline. This was possible by creating a trans-encrypting prototype as well as a reference AES-128 re-encryption implementation. Using the RSA trans-encryption implementation with an encryption step seems a relatively viable option, which leaves room for improvement if RSA hardware acceleration on the CPU were to be implemented like it has been for AES in the AES-NI instruction set. It remains to be seen if a client can decrypt at enough throughput although the 1 connection RSA trans-encryption seems to be just capable of doing a decryption at 1080p H.264 0.3MB/s throughput. The currently adopted cryptographic algorithm for the HAS pipeline AES outperformed both trans-encrypting implementations by a margin. The AES re-encryption implementation however means the use of an untrusted third party would be impossible where this would be possible for the trans-encrypting implementations. Trans-encryption increases the security of the DRM pipeline with a non insurmountable increase of overhead.

Acknowledgments

I would like to thank the TNO medialab, especially Oskar van Deventer and Thijs van Veugen, for their tutelage and guidance during this project.

References

[1] Rivest, Ronald L and Shamir, Adi and Adleman, Leonard. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[2] Kahn, David. *The codebreakers*. Weidenfeld and Nicolson, 1974.

[3] ElGamal, Taher. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[4] Damgård, Ivan and Jurik, Mads. A generalisation, a simpli. cation and some applications of Paillier's probabilistic public-key system. In *International Workshop on Public Key Cryptography*, pages 119–136. Springer, 2001.

[5] Stockhammer, Thomas. Dynamic adaptive streaming over HTTP–: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144. ACM, 2011.

[6] Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats. Standard ISO/IEC 23009-1, International Organization for Standardization, Geneva, CH, May 2014.

[7] Information technology – MPEG systems technologies – Part 7: Common encryption in ISO base media file format files. Standard ISO/IEC 23001-7, International Organization for Standardization, Geneva, CH, February 2016.

[8] Daemen, Joan and Rijmen, Vincent. AES proposal: Rijndael. AES Algorithm Submission. *National Institute for Standards and Technology (NIST)*, September 1999.

[9] Veugen, Peter and Van Deventer, Mattijs Oskar and Niamut, Omar Aziz. Secure distribution of content, March 28 2013. European Patent Office PCT/EP2012/067577.

[10] Will Glozer. WRK Modern HTTP benchmarking tool. `https://github.com/wg/wrk`, 2017.

[11] Banga, Gaurav and Druschel, Peter. Measuring the Capacity of a Web Server. In *USENIX Symposium on Internet Technologies and Systems*, pages 61–71, 1997.

[12] Mosberger, David and Jin, Tai. httperfa tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37, 1998.

[13] Batina, Lejla and Lano, Joseph and Mentens, Nele and Ors, Siddika Berna and Preneel, Bart and Verbauwhede, Ingrid. Energy, performance, area versus security trade-offs for stream ciphers. SASC - The State of the Art of Stream Ciphers Brugge, Workshop by ECRYPT Network, October 14 2004. `http://hdl.handle.net/2066/127475`.

[14] Barkan, Elad and Biham, Eli and Keller, Nathan. Instant ciphertext-only cryptanalysis of GSM encrypted communication. *Advances in Cryptology-CRYPTO 2003*, pages 600–616, 2003.

[15] Lu, Yi and Meier, Willi and Vaudenay, Serge. The conditional correlation attack: A practical attack on Bluetooth encryption. In *Crypto*, volume 3621, pages 97–117. Springer, 2005.

[16] AlFardan, Nadhem J and Bernstein, Daniel J and Paterson, Kenneth G and Poettering, Bertram and Schuldt, Jacob CN. On the Security of RC4 in TLS. In *USENIX Security Symposium*, pages 305–320, 2013.

[17] Diffie, Whitfield and Hellman, Martin. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[18] AE Okeyinka. Computational speeds analysis of RSA and ElGamal algorithms on text data. In *Proceedings of The World Congress on Engineering and Computer Science*, pages 115–118, 2015.

[19] Abdullah, Mohammed Najm and Al-Chalabi, Atheer Marouf. Performance Assessment of RSA, ElGamal and Proposed DHOTP for File Security in Pervasive Computing Environment. *International Journal of Advanced Research in Computer and Communication Engineering*, 5(1), December 2016.

[20] Mini Malhotra and Aman Singh. Study of various cryptographic algorithms. *IJSER*, 1(3):77–88, 2013.

[21] Paweł Piotr Przeradowski. Japronto. `https://github.com/squeaky-pl/japronto`, 2017.

[22] Kazuho Oku, Tokuhiro Matsuno, Daisuke Murase and Shigeo Mitsunari. PicoHTTPParser. `https://github.com/h2o/picohttpparser`, 2017.

[23] Yury Selivanov. uvloop. `https://github.com/MagicStack/uvloop`, 2017.

[24] Dan Boneh et al. Twenty years of attacks on the rsa cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

[25] Fadhil, Heba Mohammed and Younis, Mohammed Issam. Parallelizing RSA algorithm on multicore CPU and GPU. *International Journal of Computer Applications*, 87(6), 2014.