2016

Accelerating NTRUEncrypt for in-browser cryptography utilising graphical processing units and WebGL

Dajne Win
Security & Forensics Research Group, Auckland University of Technology, dwin@aut.ac.nz

Seth Hall
Security & Forensics Research Group, Auckland University of Technology, shall@aut.ac.nz

Alastair Nisbet
Security & Forensics Research Group, Auckland University of Technology

DOI: 10.4225/75/58a6a626b43ed
Win, D., Hall, S., & Nisbet, A. (2016). Accelerating NTRUEncrypt for in-browser cryptography utilising graphical processing units and WebGL. In Johnstone, M. (Ed.). (2016). *The Proceedings of 14th Australian Information Security Management Conference, 5-6 December, 2016, Edith Cowan University, Perth, Western Australia.* (pp.60-66).

ACCELERATING NTRUEncrypt FOR IN-BROWSER CRYPTOGRAPHY UTILISING GRAPHICAL PROCESSING UNITS AND WEBGL

Dajne Win, Seth Hall, Alastair Nisbet
Security & Forensics Research Group, Auckland University of Technology,
Auckland, New Zealand
dwin@aut.ac.nz, shall@aut.ac.nz, anisbet@aut.ac.nz

Abstract

One of the challenges encryption faces is it is computationally intensive and therefore slow, it is vital to find faster methods to accelerate modern encryption algorithms to keep performance high whilst also preserving information security. Users often do not want to wait for applications to become responsive, applications on limited devices such as mobiles often compromise security in order to keep execution times quick. Often they use algorithms and key sizes which are not considered cryptographically secure in order to maintain a smooth user experience. Emerging approaches have begun using a devices Graphics Processing Unit (GPU) to offload some of the computational burden from the Central Processing Unit (CPU) in an effort to parallelize and accelerate the encryption algorithms. Programming for a GPU often involves the use of CUDA or OpenCL programming, however these approaches are platform dependant. This research focuses on utilizing a GPU to perform in-browser cryptography using WebGL and JavaScript. This allows any GPU-enabled device capable of launching an OpenGL compatible browser to perform GPU accelerated cryptography. A GPU based implementation of the NTRUEncrypt algorithm was created and tested against a CPU based version on a range of hardware devices with results, challenges and limitations discussed.

Keywords
NTRUEncrypt, GPU, browser, cryptography, encryption, WebGL.

INTRODUCTION

Cryptography is one of the tools available to facilitate secure communication between two parties. Whilst cryptography provides confidentiality of communications between the two parties, it also, when applied correctly, allows for integrity, authentication and non-repudiation (Rivest 1990). One issue with cryptography on mobile devices is that it is computationally expensive meaning when high speed networking is utilised the encryption and decryption of messages on the device can become a bottleneck for the message exchanges, slowing down the communication exchange.

Whilst developments in cryptography has seen greatly improved security of cryptography with more sophisticated and robust encryption algorithms developed in the last few decades, the speed of encryption and decryption has suffered from the increased complexity of the cryptographic processes (Kapoor, Pandya et al. 2011). The Central Processing Unit (CPU) of a device is fixed hardware that must handle complex calculations for the processing of many functions on the devices. The design of the CPU on mobile devices is deliberately conservative as it must be small, consume little battery power and produce minimal heat. It therefore tends towards being designed to provide high speed processing but at its limit of available processing power. This leaves little available processing power when complex additional computations are required for cryptographic processes.

The GPU on a mobile device is required to deal with processing of graphical images which may at times require only a small percentage of the available processing power that the unit is capable of handling. This, along with the multiple processes within a single GPU mean that much additional processing power is available but generally not utilised. The ability to direct this additional processing power towards cryptographic requirements therefore has considerable benefits in permitting the most secure cryptographic algorithms to be utilised at high speed for message exchanges (Harrison and Waldron 2010). Whilst the idea of utilising the GPU of a device for processing of non-graphical requirements is not new, the restrictions that have been imposed by previous designs means that only a select few devices are capable of implementing this technology and reaping the benefits of the additional GPU processing power. This paper describes the development of a GPU accelerated NTRU Encrypt algorithm used for in-browser cryptography. It uses the WebGL and Three.js libraries for the purpose of trying to utilise the algorithm on any browser enabled device.

STATE OF THE ART

The Data Encryption Standard ratified in 1977 provided a robust and reliable symmetric encryption algorithm suitable for public and government use. With only 56 bit encryption permitted to comply with the contemporary laws restricting encryption to this key size, Data Encryption Standard (DES) remained suitable until technology surpassed the protection afforded by such a small key length. Triple DES (3DES) acted as an intermediary fix to provide significantly stronger encryption without the need for an entirely new algorithm. However, 3DES was computationally expensive and with security weakening as computer processing increased rapidly, the Advanced Encryption Standard (AES) was developed in 1998 as a result of a competition for a replacement for DES and 3DES (Daemen and Rijmen 2003). Both 3DES and AES are commonplace and whilst AES is much preferred, 3DES remains as the encryption algorithm of choice for many banking applications. AES is computationally expensive which for computer applications is suitable but for mobile devices with much less processing power can slow message exchanges in some circumstances.

In 2015, National Institute of Standards and Technology (NIST) announced that it could no longer support continued use of the 2 key 3DES version and recommended utilising 3 key 3DES or move to AES (Barker and Barker 2004). The banking industry has been slow to respond and 2 key 3DES remains in many banking communications, however the move to AES seems inevitable as more and more research into the insecurity of DES in all varieties appears.

The need to accelerate AES and other encryption processing on these devices has seen some improvements through new processes but so far all restrict the new processes to specific hardware. In 2007, experiments with accelerating AES by utilising a GPU showed significant improvements with bulk processing of AES (Harrison and Waldron 2007). The same year saw further experiments with AES and DES which achieved improvements of over 100% throughput with a GPU compared to a high performance CPU (Yang and Goodman 2007). In both of these experiments the limiting factor for improved processing was found to be the memory bandwidth rather than the GPU processing units on each card as each card required the same number of memory fetch units. In 2012 experiments by Chesebrough and Conlon showed a significant increase in cryptographic processing of AES-NI of 5 times greater than previous experiments (Chesebrough and Conlon 2012).

The move to parallel computing has seen great benefits in processing speed but at the expense of heat generation (Oancea, Andrei et al. 2014). Technology has moved rapidly from the original 8086 processer with 29 000 transistors to the Intel core i7-920 released in 2014 with 731 000 000 transistors to 1.9 billion transistors by 2015, all within approximately the same sized footprint of the CPU (Kowaliski 2015). Whilst multiple cores have somewhat alleviated the heat generation problems, the computing power requirements have increased at a greater rate than development so that higher performance from software remains ahead of the CPU development (Oancea, Andrei et al. 2014).

The area for greatly improved processing power at little cost is in Graphical Processing Unit utilisation as a defacto CPU. Nvidia GPUs utilise Single Instruction Multiple Data (SIMD) stream architecture which has the advantage over Single Instruction Single Data (SISD) of providing natural parallelism and therefore processes data simultaneously (Nvidia 2008). Nvidia's modification of SIMD is termed Streaming Multiprocessors (SM) and places emphasis on ease of development by reducing the complexity of the design. This has seen significant research in the area of GPU accelerated cryptography tied to Nvidia's hardware because of the simplicity with which developers can utilise development platforms such as Compute Unified Device Architecture by NVidia (CUDA), (Yang and Goodman 2007).

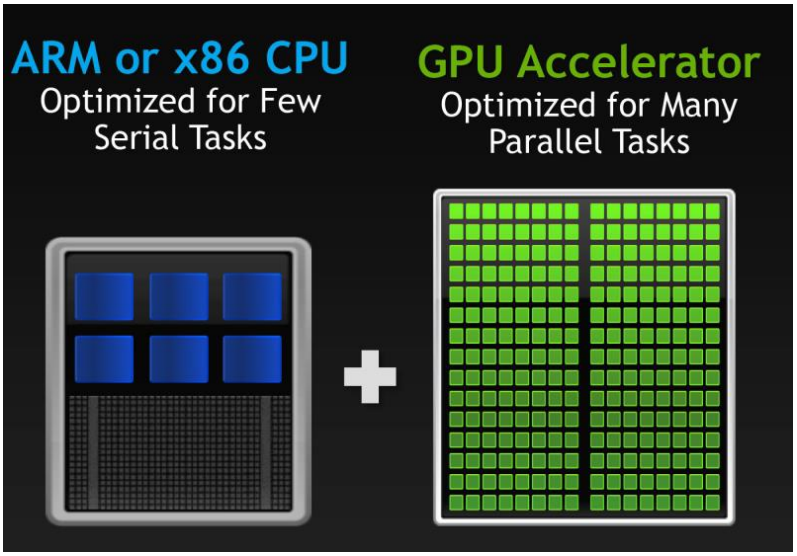

Figure 1: CPU v GPU processors (Nvidia, 2008)

As illustrated in Figure 1, the main difference in processing of a CPU compared to a GPU is that a CPU may consist of several cores with a control bus for communication meaning that the CPU does not perform in a truly parallel manner. The GPU is made up of many floating point units, with each unit dedicated to performing arithmetic tasks and when combined in a truly parallel manner produce a truly SIMD architecture which is considerable faster than the CPU. In 2008 an Nvidia 8800 GTX was utilised as it allowed for the use of DirectX 10, CUDA and a 32 bit integer processor (Harrison and Waldron 2009). They noted the necessity of taking into account different memory architectures to improve performance as whilst there is some local storage available, most is off-chip making transfers comparably slower. The competing architecture is OpenCL (Open Computing Language) and there has been significant research into utilising GPU's of each type but specific to the type being accelerated. This emphasis on writing software to accelerate the Nvidia architecture or the OpenCL architecture has meant that developments work for one technology but not the other.

Accelerated programming has become a necessity not just because of the increased speed of computing applications but because of the greatly increased expectations of the users. In 1994, Nielson's research found that 0.1 seconds of latency after performing a task such as selecting a key on a keyboard or dragging the mouse pointer across a screen could be considered instantaneous. Similar research by Ritter et al in 2015 found that 300 milliseconds was acceptable as a maximum latency with 170 milliseconds required for more urgent tasks (Ritter, Kempter et al. 2015).

The issue with acceleration of processing by utilising the GPU has potential to be overcome by the implementation of acceleration in JavaScript. All modern web browsers support the use of JavaScript (Flanagan 2011) and with the introduction of HTML5 JavaScript has moved from a simple scripting language to an efficient programming language. With a doubling of Internet connected devices occurring every two years (Gartner 2015) , the motivation to improve processing speeds on mobile devices becomes very apparent. An early attempt in 2012 to implement JavaScript on a GPU utilising OpenCL rather than the Nvidia architecture on an Intel i7 processor proved unsuccessful at accelerating the processing but did prove that it was technically possible to at least do so (Nicholls 2012). One benefit of a successful implementation is that client side acceleration of processing functions could occur efficiently, alleviating some of the burden of the processing from the servers. The first investigation into utilising the GPU for accelerating cryptographic functions using JavaScript occurred in 2006 (Cook and Keromytis 2006). Their experiments targeted the AES algorithm but problems were experienced in the areas of a lack of modular arithmetic, unsigned integers, branching and large integers. Yang and Goodman directly furthered the experiments and gained some positive results primarily because of the development of the technology in the 1 year period (Yang and Goodman 2007). In that time, branching and unsigned integers no longer proved difficult to handle but large integers remained a problem. Two years later, experiments of a similar nature but with asymmetric cryptography also showed some positive success but with some limitations (Harrison and Waldron 2009). Their research involved accelerating 1024 bit RSA encryption and they were able to show higher throughput and decreased latency by utilising the Nvidia GPU. Since 2010, little advancements have been made in accelerated cryptography utilising the GPU although hardware advances have made the area far more promising for success.

This issue with all of these previous experiments remained that they were all targeted to a particular type of GPU architecture, either Nvidia or OpenCL but not both. To be universally beneficial, what is clearly needed is software in JavaScript that will accelerate cryptography in both architectures. The following section outlines the research design for the software development.

RESEARCH DESIGN

This research used an experimental research design methodology which allows for control over the variables defined in the experiment, allowing more accurate results to determine relationships between sets of data. The purpose of running experiments with software that is developed to improve performance over currently available software is to show that the new scheme has benefits over previous schemes. A performance benchmark of what is currently available is one method of showing that improvements have been made with the new scheme by way of comparison between the older and the new scheme. Crypto++ has been chosen as the cryptographic scheme to be used in the comparison. Crypto++ is an open source library of several cryptographic algorithms and the version that is used in this research provides for AES encryption. Utilising the JavaScript software and RSA (Rivest, Shamir et al. 1978) cryptography as well as Elliptic Curve Cryptography (ECC) (Koblitz 1987). The JavaScript is run encrypting NTRUEncrypt and the resulting data used to compare against previous schemes. The hardware utilised for the testing can be divided into 3 distinct types, mobile, desktop and laptop, and server. These are shown in the following tables (Table 1 and 2) as well as the CPU, GPU, RAM and operating systems used.

Table 1: Mobile platforms tested

	Samsung Galaxy S2	ASUS Zenfone 2	Samsung Tab 10.1	Nvidia TK1
CPU	Dual Core 1.2 GHz Cortex A9	Quad Core 2.3 GHz Intel Atom Z3580	Dual Core 1 GHz Cortex A9	Quad Core ARM Cortex A15
GPU	Mali 400 GPU	PowerVR G6430 GPU	Nvidia Tegra 2 T20 ULP Geforce	Nvidia Kepler, 192 CUDA Cores
RAM	1 GB	4 GB	1 GB	2 GB
OS	Android 4.1.2	Android 5.0	Android 3.1	Ubuntu Desktop 14.04 LTS

Table 2: PC desktop platforms tested

	Custom Desktop	HP Desktop	Toshiba Thinkpad
CPU	Quad-Core 3 4 GHz Intel i5 3570k	Quad-Core 3 2 GHz Intel i5 4570	Dual-Core 2 53 GHz Intel Core 2 Duo T9400
GPU	Nvidia 660 GTX	AMD Radeon R7 360	Intel GMA 4500MHD
RAM	16 GB	16 GB	2 GB
OS	Windows 10	Ubuntu Desktop 14 04 LTS	Lubuntu 14 04

Initial tests were run to check that the systems were working as expected and results were being correctly recorded. Once this was satisfied, the experiments could begin.

TESTING & RESULTS

This research focused on utilizing the GPU in an attempt to optimize NTRUEncrypt (Hoffstein, Pipher et al. 1998) algorithm for in-browser cryptography. In order to target a variety of different devices, a Javascript version of NTRUEncrypt was developed using Three.js, which is a cross-browser library using WebGL and is primarily used for computer graphics. The experiments compared both Javascript CPU and GPU implementations of NTRU encrypt to test whether it could be beneficial to utilize the GPU for in-browser cryptography. Average throughput results after 500 iterations for each version of NTRU on the various platforms were recorded and compared. Porting an algorithm to GPU brings a number of challenges in order to port the algorithm to be more suitable for the GPU pipeline such as reducing code branching, fixing float values to between 0-1 within the GPU, and floating point precision errors. There is also limits on the buffer sizes and number of uniform values that are passed into the each shader. NTRUEncrypt is was used as it is considered to be fast compared to other asymmetric encryption algorithms (Hermans, Vercauteren et al. 2010) with comparable security settings and “quantum resistant” (Perlner and Cooper 2009), making it an ideal choice for future cryptography systems.

The algorithm was first tested with small set NTRU parameters (N=11) and progressing upwards to higher sets (N=1499, ees1499ep1) which become more computationally intensive to encrypt. The following values represent NTRU parameter sets considered to be cryptographically secure (Figure 2) (Inc 2014).

	N	q	p
Moderate Security	167	128	3
Standard Security	251	128	3
High Security	347	128	3
Highest Security	503	256	3

Figure 2: Recommended NTRU parameter steps

The following GPU pipeline diagram (Figure 3) shows the parts to the implemented algorithm. The first stage of the pipeline takes in the public key and blinding values as byte arrays into the convolution shader. The output of which is then fed into the second shader where a polynomial addition modulo by the value q is performed on the plaintext to get its corresponding ciphertext.

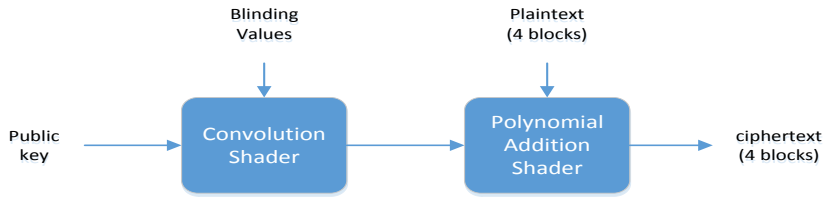

Figure 3: NTRUEncrypt GPU pipeline model

Because of the SIMD nature of GPU, the algorithm utilized vec4 operations within the GPU meaning that processing four blocks of plaintext can be converted to four ciphertext blocks with little to no extra computational cost over simply converting one block. The GPU implementation of NTRUEncrypt was tested and compared to a pure CPU implementation also implemented in Javascript and used as a benchmark comparison. The following table (Table 3) shows the results for the ees401ep1 parameter set on custom desktop.

Table 3: GPU vs CPU average throughput results in milliseconds for ees401ep1

Encryptrion Operation	NTRUEncrypt.js	NTRUEncrypt-GPU.js
Convolution	0.928	23.42
Addition	0.274	9.28
Total	2.104	230.5

Unfortunately it appears that no outright acceleration was achieved and in fact the algorithm execution time was significantly slower. This is because the nature of the algorithm for NTRU is difficult to parallelize, in particular the polynomial addition part of the algorithm. The algorithm took on average 2.1mS to encrypt one block of plaintext for the CPU implementation whereas the GPU implementation took on average 230.5mS to encrypt four blocks of plaintext held inside a vec4 buffer. Custom desktop was only able to handle ees401ep1 parameters due to the restrictions of OpenGL set on the GPU. The results are also affected by the setup time for Three.js and overhead on WebGL calling shader instances as it is simply meant to be a graphics library and not intended for high performance computing and cryptography. Furthermore hardware issues affected results due to their poor implementation of WebGL as although it is stated that it is OpenGL compliant, not all the standards are met. Furthermore the mobile devices tested also failed to perform the algorithm correctly due to lack of float buffer support.

However the following graph (Figure 4) shows results performed on the HP Elite desktop and demonstrates that, utilizing the GPU becomes advantageous when using the ees1449ep1 parameter sets for higher security levels as using a pure CPU approach becomes computationally burdensome using parameter sets over ees887ep1. Interestingly using very low spec parameters for NTRU on the GPU was slow compared to higher specification parameters, this is due to under-utilization of GPU cores. Furthermore only the original NTRUEncrypt algorithm was implemented, (Inc 2014) has suggested several enhancements to the algorithm which may be advantageous for reducing computation time for encryption/decryption on GPU.

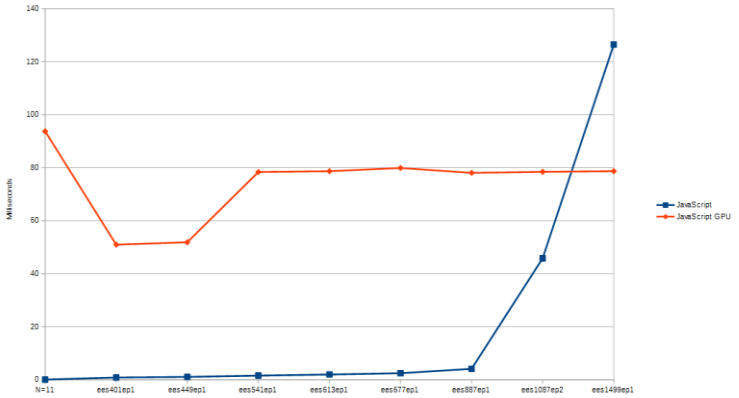

Figure 4: CPU vs GPU average throughput results on HP Elite desktop for increasing NTRU parameter sets

CONCLUSIONS

The ability to accelerate cryptography on limited devices is essential in order for applications to use encryption schemes which are considered cryptographically secure whilst still maintaining a smooth user experience. This paper looked at utilizing the GPU for in-browser cryptography using NTRUEncrypt as a case study. Not all the hardware platforms tested were able to accelerate the NTRUEncrypt algorithm, however the HP Elite Desktop was able to achieve GPU acceleration with the ees1449ep1 parameter set. Browsers poor implementations of WebGL affect performance in calling shaders for non-graphics purposes, however with the rise of GPU implementing compute shader pipelines then using GPU for non-graphics purposes could become more commonplace. Modern browsers will see the implementation of WebGL 2.1 in the near future which will have direct support for compute shaders (Jackson and Gilbert 2015). Using compute shaders is more useful for utilizing the GPU for non-graphics tasks such as high performance computing and cryptography. Future work will investigate using compute shader further by trying to improve throughput performance of a GPU-based NTRUEncrypt algorithm. A preliminary C++ implementation with the full OpenGL pipeline, utilizing compute shaders has been developed with beta results that are promising. The implementation was able to process four blocks of plaintext using the ees1499ep1 parameters in 18mS on the HP Elite Desktop. This is a significant improvement compared to the 80mS shown in this papers in-browser GPU JavaScript version. This shows that the ability to use GPU to accelerate cryptography algorithms has the potential to increase security and intelligence capabilities. However there is still a long way to go in order for it to be fully supported by browsers so it can be then used for in-browser cryptography. For now applications could still benefit from GPU acceleration by creating a direct port into the compute shader pipeline or using platform dependant alternatives such as CUDA or Open CL.

REFERENCES

Barker, W. C. and E. Barker (2004). Recommendation for the triple data encryption algorithm (TDEA) block cipher, US Department of Commerce, Technology Administration, National Institute of Standards and Technology.

Chesebrough, R. and C. Conlon (2012). Implementation and Performance of AES-NI in CyaSSL Embedded SSL.

Cook, D. and A. D. Keromytis (2006). Cryptographics: exploiting graphics cards for security, Springer Science \& Business Media.

Daemen, J. and V. Rijmen (2003). "AES Proposal: Rijndael. htt p." csrc. nist. gov/archive/aes/rijndael/Rijndael-ammended. pdf.

Flanagan, D. (2011). JavaScript: The Definitive Guide: Activate Your Web Pages, O'Reilly Media.

Gartner (2015). Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015.

Harrison, O. and J. Waldron (2007). "AES Encryption Implementation and Analysis on Commodity Graphics Processing Units." 4727: 209-226.

Harrison, O. and J. Waldron (2009). "Efficient Acceleration of Asymmetric Cryptography on Graphics Hardware." 5580: 350-367.

Harrison, O. and J. Waldron (2010). "GPU Accelerated Cryptography as an OS Service." 6480: 104-130.
Hermans, J., F. Vercauteren, et al. (2010). Speed records for NTRU. Topics in Cryptology-CT-RSA 2010, Springer: 73-88.
Hoffstein, J., J. Pipher, et al. (1998). NTRU: A ring-based public key cryptosystem. Algorithmic number theory, Springer: 267-288.
Inc, S. I. (2014). "NTRU Enhancements 1." Retrieved 1 Ocotober 2016, from https://assets.securityinnovation.com/static/downloads/NTRU/resources/NTRU-Enhancements-1.pdf.
Inc, S. I. (2014). "NTRU PKCS Tutorial." Retrieved 1 October 2016, from https://assets.securityinnovation.com/static/downloads/NTRU/resources/NTRU-PKCS-Tutorial.pdf.
Jackson, D. and J. Gilbert. (2015). "WebGL 2 Specification." from https://www.khronos.org/registry/webgl/specs/latest/2.0/.
Kapoor, B., P. Pandya, et al. (2011). "Cryptography." Kybernetes 40(9/10): 1422-1439.
Koblitz, N. (1987). "Elliptic curve cryptosystems." Mathematics of computation 48(177): 203-209.
Kowaliski, C. (2015). Intel's Broadwell-U arrives aboard 15W, 28W mobile processors.
Nicholls, J. (2012). JavaScript on the GPU.
Nvidia (2008). CUDA Programming Guide.
Oancea, B., T. Andrei, et al. (2014). "GPGPU Computing." arXiv preprint arXiv:1408.6923.
Perlner, R. A. and D. A. Cooper (2009). Quantum Resistant Public Key Cryptography: A Survey. Proceedings of the 8th Symposium on Identity and Trust on the Internet, New York, NY, USA, ACM.
Ritter, W., G. Kempter, et al. (2015). User-Acceptance of Latency in Touch Interactions. Universal Access in Human-Computer Interaction. Access to Interaction. M. Antona and C. Stephanidis, Springer International Publishing. 9176: 139-147.
Rivest, R. L. (1990). Cryptography. Handbook of Theoretical Computer Science. J. V. Leeuwen, Elsevier. 1: 717-755.
Rivest, R. L., A. Shamir, et al. (1978). "A Method for Obtaining Digital Signatures and Public-key Cryptosystems." Commun. ACM 21(2): 120-126.
Yang, J. and J. Goodman (2007). "Symmetric Key Cryptography on Modern Graphics Hardware." 4833: 249-264.