The Effects of Crypto-Hardware on Low-end Internet of Things Performance

DR.PASALA RAJA PRAKASH RAO[1], G.MANGA RAO[2],
PROFESSOR[1], ASSISTANT PROFESSOR[2],
DEPARTMENT OF ECE
PBR VISVODAYA INSTITUTE OF TECHNOLOGY AND SCIENCE::KAVALI

Abstract

In this article, we quantify the performance effect of crypto-hardware by doing a complete resource analysis of commonly used cryptographic primitives on a variety of commercially available IoT systems. The foundation of this study is the recently developed crypto-subsystem of the RIOT IoT operating system, which enables cross-component crypto support. I Hardware-based cryptography provides far better performance than software-based cryptography, as shown by our tests; this is critical for node longevity. However, moderate memory enhancements are the norm. (ii)There are several factors that influence resource efficiency, including hardware variety, driver design, and software implementations. Even if they are inefficient for symmetric crypto operations, external crypto-chips do provide secure write-only memory for private credentials—something that is lacking in many other systems.

Introduction

The foundation of the Internet of Things is security (IoT). Crypto-operations are necessary for data privacy, integrity, and accessibility, yet they are typically inefficient and in contradiction with device limitations. Software upgrades, access control systems, and data encryption all need crypto-operations. Cryptographic primitives, including potential crypto-extensions, need to be significantly optimised to offer practical security in the low-end IoT.

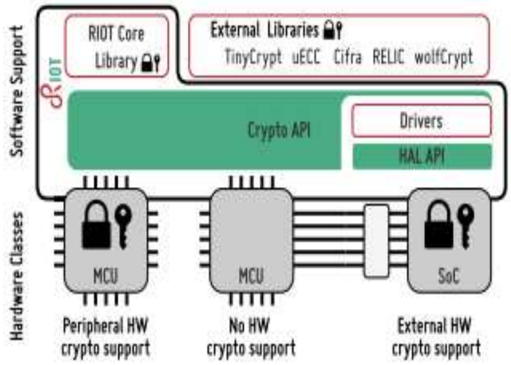

Figure 1. The software support layer of RIOT integrating crypto-peripherals, external crypto-devices, and cryptolibraries using a common crypto API.

a formidable obstacle, since hardware support is all over the place, from incomplete to fully-functional implementations of widely-used algorithms like AES. The restricted IoT security choices shown in Figure 1 are the most often used ones. There are three types of crypto-related hardware: I microcontrollers with built-in crypto-peripherals, (ii) external crypto-devices that interface to the microcontroller through a communication bus, and (iii) cryptographic software lib raries that attempt to deal with embedded limitations. For reasons of mobility, software libraries avoid using crypto hardware, while manufacturer SDKs (on bare metal) decrease freedom in the direction of a vendor lock-in. The use of an operating system (OS) is becoming more common in IoT deployments because it allows applications to remain portable while providing near-optimal hardware support via an abstraction layer. One of the primary goals of this study is to quantify the resource benefit from making heterogeneous hardware components consistently available to both crypto-libraries and apps. Until recently, there has been little availability of crypto-hardware with platform independence at the IoT system level. We propose that an Internet of Things operating system should have standardised APIs that allow access to any accessible hardware without compromising on speed or features. We will report on the many difficulties that arise from using disparate hardware ideas and having limited resources, and how they might be overcome by exchanging software for hardware without reducing productivity. The following are the paper's contributions:

We add our crypto subsystem to the IoT operating system RIOT (2) and discuss design considerations for integrating various crypto-drivers. A quick recap of the hardware crypto systems available today (3). The third section compares the efficiency of five software libraries (4), the fourth section examines the implementation of rudimentary symmetric and asymmetric cryptography on four hardware platforms, and the sixth section discusses more complex Elliptic Curve Cryptography (ECC). Our findings suggest that hardware is not always the best option. 4.Optimization opportunities are revealed by a comprehensive benchmarking study of vendor drivers integrated into the system (7). All of our software may be found at https://github.com/ inetrg/EWSN-2021.

RIOT's Cryptographic Functions

Here, we present the architecture and implementation of a crypto-subsystem that may compare well across different libraries and operating systems. The RIOT [8] open-source operating system for low-end IoT microcontrollers serves as the foundation for our implementation. We choose RIOT because it is portable across several architectures (8-bit, 16-bit, and 32-bit CPUs), has a scheduler that allows for set priority and pre-emption, manages power consumption well (see RFC 3636), and offers a robust hardware abstraction layer. In today's systems, cryptographic functions are often implemented as software [18]. Alternately, third-party libraries may be integrated through the package system. Wolf Crypt [43], an embedded library for symmetric and asymmetric crypto, Ciera [10], which implements common building blocks for symmetric crypto, Tiny Crypt [20], and micro-etc(uECC) [21], both particularly minimising memory, and Relic [4], which contributes a comprehensive list of symmetric and asymmetric cryptographic schemes with particular support for many elliptic curves, are all included in RIOT. For this reason, these external libraries have not been ported to use any APIs provided by the operating system. Our design idea is broad enough to include these elements and can be easily adapted to accommodate more hardware platforms and libraries.

Important Things to Think About With the Design

The ability to use cryptographic hardware is a feature of modern operating systems. A driver is software that operates the device and implements an operating system-independent application programming interface (API). Vendors supply a library in this paper's five example use cases so that users may perform low-level operations. Design considerations for integrating these and future cryptographic components are presently under discussion.

Interoperability with Vendor Drivers

There is a broad range of functionality across vendor drivers. We believe, however, that these implementations should be used to take advantage of specialised vendor expertise, testing, and the possibility of long-term maintenance. During the firmware compilation process, RIOT's package subsystem will clone, compile, and link to external repositories. This makes it such that updates to third-party programmes don't need to be made inside the OS itself. To facilitate the incorporation of third-party code into the underlying system, we offer software wrappers and implement vendor libraries as RIOT packages. It is important to note that, as we shall demonstrate in Section 7, vendor libraries may not always provide optimal performance since they are often developed in a generic manner.

Abstraction from Context

In order to work, cryptographic algorithms rely on a secret state (con text struct). Each instance of a driver receives its own, and how much is allocated for it relies on the revealed state of the vendor's implementation, which often contains its own set of hardware-specific components. When interacting with the operating system, a context struct must abstract vendor-specific details and implement standard OS interfaces. Because of this, all drivers create a standard context struct that incorporates OS-specific and vendor-specific features. Since the context struct varies among backends, API consumers should avoid deriving from it. Due to this design choice, many backends cannot perform the same task simultaneously. We propose three reasons why this is typical in Internet of Things deployments: Real-world IoT firmware is purpose-built for a single task, hence single-core OSes are not designed for parallel processing I Because of this, we do not anticipate that crypto-operations will be heavily parallelized. (ii) Constrained IoT devices have limited computational and memory resources. We do this by abstracting the context. Hardware operations that are performed in succession are already much faster than software solutions, and the performance of crypto-peripherals boosts software solutions by an order of magnitude (see Section 5).

CombiningSecurity-Related Components

There are a wide range of crypto-hardware capabilities among the 117 microcontrollers and 208 boards now supported by RIOT. We provide I am hardware-agnostic API and (ii) the dynamic configuration of the crypto subsystem to enable the construction and usage of crypto-based apps without prioritising the hardware setup. To communicate crypto hardware features to the build system, we use a feature model. When applicable, our method selects and builds hardware features and also offers a more robust configuration interface. As long as it is accurately modelled, this generic method can handle any hardware component.

Modular Structure

User-facing APIs are introduced in two flavours by our layered approach to interacting with various crypto backends: A single AES block encryption, for example, is a low-level function that may be accessed directly using the basic cryptographic API. For example, you may set up AES to work in Cipher Block Chaining (CBC) or Electronic Code Book (ECB) mode with the use of the API for cryptographic modes. Because each backend is modelled as a separate module, the build system can pick and choose which one to use based on its specifications. The three types of hardware crypto-acceleration that can be supported by a backend module (or driver) are I full hardware acceleration, (ii) partial hardware acceleration, and (iii) no hardware acceleration. Devices in the periphery, or those used externally, that provide complete hardware support for a cryptographic mode form the foundation of this tier (e.g., AES CBC). When just the most fundamental cryptographic operations are supported by the hardware, we have reached the second level. Software implementations of the operation mode (such as CBC) need access to standard cryptographic primitives (e.g., AES block encryp tion). The software part, on the other hand, doesn't care what kind of encryption or hash is being used. The third difficulty setting simulates a situation in which necessary hardware acceleration units are unavailable. We've provided an abstraction of the cryptographic API that makes it easy to use with a variety of backend implementations.

Constructing a Measuring Device

Oversight of the Environment

The hardware and its capabilities used for testing are summarised in Table 1. In our tests, we avoid using outdated algorithms and implement them on both cutting-edge (nRF52840 and EFM32) and more dated (MKW22D) microcontrollers equipped with peripheral crypo-acceleration. The nRF52840 and EFM32 are two examples of the new generation of devices with advanced crypto-peripherals that were designed for easy deployment in common scenarios. In addition, an external crypto-chip (ATECC608A) is used, linked to the system through the I2C bus. The ATECC608A family of external security components is impervious to side channel assaults. Except for the MKW22D, all platforms have a TRNG that is in accordance with NIST standards (cf., [22] for background on embedded ran dom number generation). The ATECC608A is equipped with a cryptographically secure pseudo-random number generator (CSPRNG) that is seeded by a genuine random component. Both the EFM32 and the MKW22D use hardware hashing to implement software-assisted HMAC SHA-256. Both the nRF52840 and the EFM32 have hardware support for several cypher modes, unlike the MKW22D and the ATECC608. With the exception of MKW22D, all other systems support ECC. For cryptographic operations, both the nRF52840 and the EFM32 provide protected key registers, whereas the MKW22D provides a timer for each secure register. The ATECC608A can keep secrets in its sixteen slots of non-volatile, write-only memory. To prevent unwanted access, keys are produced and stored on an external device, which is also responsible for erasing the keys upon detecting tampering.

The Effects of Putting New Software in Place

We performed RIOT on the nRF52840 platform and compared the results of several software implementations of SHA-256 and AES-128 offered by commonly accessible crypto-libraries (Tables 2 and 3). In both situations, an input vector of size one internal block was used in conjunction with the cryptographic algorithm (i.e., 64 Byte for SHA-256 and 16 Byte for AES 128). In order to compute a SHA-256 digest, Relic, Tiny Crypt, and Cifra need between 190 and 210 s for an init-update-final sequence. However, Cifra's quicker update speed comes at the cost of a longer finalisation time due to the usage of a second copy of the hash value. With RIOT Core's state update requiring many modulo operations, the time required is 20 s longer (FIPS PUB 180-4). In order to work with 32-bit arithmetic, RIOT provides several endianness conversions. Although it requires more memory, wolfCrypt's implementation is extremely efficient because to its unrolled mixing loop. Turning off this optimization decreases ROM needs by 500 Bytes at the expense of an additional 100 ms of processing time for update and final. All implementations utilise around the same amount of stack and have roughly the same-sized contexts, with the exception of Relic, which needs 400 Bytes more stack for a global array containing initial hash state values. AES-128 is very sensitive to variations in software implementation. RIOT, wolf Crypt, and Relic all have quick initialization times; it only takes roughly 3 s to set up the state and the AES key length. Rather of providing a dedicated API call for initialization, Tiny Crypt takes care of this internally. Ciera, on the other hand, has the key schedule (FIPS PUB 197) built in during its lengthy start-up process (up to 50 s).

While RIOT performs AES key expansion on each encrypt/decrypt call, wolf Crypt, Relic, and Tiny Crypt provide a separate API to initiate the process. Except for Cifra, the key expansion overhead is included in the encrypt and decrypt columns in Table 3. Due to the extra key inversion required during decryption, encryption is 1.5-3 times quicker than decryption. [12] For a single block, RIOT is the quickest implementation, followed by wolf Crypt and Relic. The implementations rely on look-up tables (T-tables) that have already been constructed in advance to increase performance on 32-bit systems. The default approach in Cifra (unprotected) and Tiny Crypt is based on a substitution table (S-box). Unexpectedly, Cifra's (unprotected) S-box implementation scales similarly to the T-table technique, but Tiny Crypt is four to ten times slower. This is because of the redundancy introduced by keeping multiple copies of the state in both the internal and externally given locations and by the need of periodically purging the internal memory. Cifra (w/ protection) offers countermeasures by default, which increase the runtime by a factor of 100, since lookup table implementations are susceptible to side channel assaults [7, 41], particularly cache attacks.

Acceleration of Cryptographic Processes using Some Very Simple Hardware

We next use the crypto hardware we covered in Section 3.1 to evaluate how quickly certain cryptographic operations can be performed in hardware vs. how quickly they can be performed in software. The same platforms are used to acquire RIOT core software results, but this time the crypto-hardware is disabled.

Computer Time

Data lengths of 32 bytes and 512 bytes were processed at different rates, as shown in Figure 2. For the CBC mode, we generate both the 128-bit AES key and the initialization vector at random. A random 256-bit key is used to initialise the HMAC SHA-256. Experiments with 512-bit keys were also conducted, as is sometimes suggested. Inputs are kept to a minimum. Ini tribalization, encryption, and decryption times for AES ECB/CBC and SHA-256 hashing are less than 70 s on both the nRF52840 and the EFM32 for short inputs (Fig. 2(a)). A more complex algorithm, HMAC SHA-256 requires at most 250 s on both platforms and is used for repeated in ternal hash computations. Due to the short input sequence, hash updates are small for all configurations. In this case, final is the trigger that causes the update function to gather 64 Byte of data (SHA-256 internal state) before beginning a block operation. The MKW22D requires very little extra time or effort to perform any task. On that system, AES CBC encryption is slower than decryption due to the software chaining of hardware-accelerated AES blocks. In order to prevent overwriting the input buffer, another copy is made before encryption but before decryption. When implemented in hardware, cyphers gain more— a factor 4–6—over software than hashes (factor 2–4). (Factor 2–4). A comparison of software and hardware measurements for the EFM32 shows the particular power of that platform. It opera ates at minimal cost using hardware accelerated operations, in contrast to software, for which it performs slower than nRF52840 and MKW22D, since it operates at lowest CPU frequency. The extra key inversion makes software-based AES ECB/CBC decryption twice as slow as encryption (see Section 4). This extra work is not needed on the physical machine. As can be seen in Figure 2(a), the ATECC608A is two orders of magnitude slower than the other platforms. There are two factors contributing to this expense. To begin, the vendor library keeps track of the power state of the device and wakes it up just before each operation. Second, the microcontroller must send and receive copies of control instructions and data through the I2C connection. The longer AES initialization takes is proportionate to the time it takes to send the encryption and decryption key to the device. Due to the fact that AES-128 encryption of 32 Byte takes two block operations, the transmission of which adds an overhead, cypher and hash-based methods have a larger performance gap on the ATECC608A compared to crypto-peripheral and software support.

Hardware Enhanced Error-Correcting Code

We provide studies of both hardware and software elliptic curve encryption systems. When evaluating the hardware's performance, we take the nRF52840's peripheral crypto acceleration and the ATECC608A's external crypto chip into account. Relic, a library packed with features, and quick, a basic library with a focus on optimization, are both used to monitor software performance on the same platform. By default, Relic is configured to utilise a precomputation table for scalar multiplication, which significantly boosts the application's speed while running. The optimal compromise between performance and size is achieved by deploying uECC with the default optimization level of 2. We evaluate keypair creation, signature generation and signature verification (ECDSA), and the development of a shared secret, based on previous key exchange, using the NIST P-256 elliptic curve with 256-bit sized keys that is supported by all hardware and software systems (ECDH). The maximum size for a secret key or message digest is 32 bytes (256 bits), and signatures are calculated using 32 bytes. Hardware accelerate Rs employ a built-in TRNG for key pair generation and signature. The CSPRNG used for our software metrics is based on the Secure Hash Algorithm (SHA-256), and it is seeded. As an additional measure, we set up both libraries to make use of a hardware generator; nonetheless, the benefit is still very little. We did not include these kinds of trials since their findings did not provide any new information.

The Consequences of New Driver Features

Provided by Vendor and Shared Network Access

The EFM32 (V. PG12) has two independently operable crpyto-peripherals. In a single-core system, this concurrent capability is handled through a driver API that must be asynchronous to properly arrange peripheral access. The

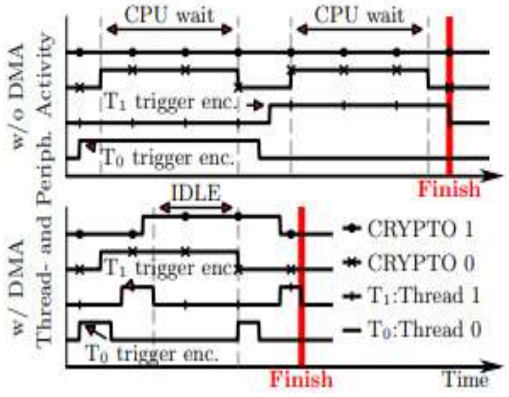

Figure 2 . Qualitative comparison of thread and cryptoperipheral activity with (bottom) and without (top) CPU offloading using DMA.

However, vendor implementations of crypto-operations stall the CPU. The results of our test application using the vendor driver are shown in Figure 1 (top). We launch two identically prioritised threads, one of which will encrypt data periodically, while the other will decrypt it. T0 initiates encryption on the CRYPTO0 subsystem. Until the hardware is finished, T0 has taken over the CPU. This frees up CRYPTO 0 for use once again, at which point T1 is automatically triggered to initiate an encryption.

Due to the vendor driver's emphasis on parallelism, CRYPTO 1 is never utilised. We use Direct Memory Access (DMA) to offload the CPU in our asynchronous driver implementation. Both the input data and the encrypted output data are sent between the device and the peripheral registers using direct memory access (DMA). The development of our test application with the improved driver is seen in Figure 6 (bottom). To free up the CPU while the peripheral does its thing, pressing T0 activates CRYPTO 0. When T1 is scheduled, it forces CRYPTO 1 to begin encrypting, freeing up processing time. It's important to take note of the fact that the CPU is currently idle while both of the peripherals work in parallel. The OS may plan other activities or enter a power-saving mode during such period. Every thread receives a message letting it know when the auxiliary jobs are done.

Synopsis and Future Prospects

To the best of our knowledge, this work presents the first systematic comparison of numerous symmetric and asymmetric cryptographic algorithms, both in hard- and software implementations, and consistently assessed on a wide range of resource-limited, widely-deployed IoT devices. We demonstrated extensive system benchmarks for a sample set of crypto-peripherals and an external security device in order to shed light on the compromises that must be made in order to provide safe crypto-hardware support on a general-purpose operating system for resource-limited devices. Among our findings are: In terms of speed and power consumption, crypto-peripherals are superior than software. The advantage grows as the duration of the input is increased. This extends the period that a node may function. Unfortunately, drivers add unnecessary memory use. (ii) The scalability of crypto-hardware in regards to con text sizes and stack usage is on par with that of crypto-software. Adding more complexity to a device will naturally cause more overhead. Despite being rather sluggish for symmetric crypto-operations, external crypto devices provide significant performance gains for asymmetric crypto. Cryptographic operations are possible on very limited systems due to their low memory requirements. Additionally, a suite of hardware-based side-channel countermeasures offerfurther protection against assaults. The I2C protocol adds a potential vulnerability to the system. Extra caution is needed when dealing with crypto-drivers (iv). Numerous vendor implementations were discovered to have significant optimization potentials. In addition, a flexible environment with many abstraction and software layers is required for various degrees of hardware crypto capability. The OS provides this, which helps with code portability and reuse as well as taking use of hardware capabilities. We believe our findings may serve as a basis for future action to avoid performance issues.

References

[1] M. Al-Zubeida, Z. Zhang, and J. Zhang. Efficient and Secure ECDSA Algorithm and its Applications: A Survey. Int. Journal of Communecation Networks and Information Security (IJCNIS'19), 11(1), 2019.

[2] Apache Software Foundation. Contiki-NG: The OS for Next Generation IoT Devices. https://github.com/contiki-ng/contiki-ng, last accessed 10-11-2020.

[3] Apache Software Foundation. Apache Mynewt. https://mynewt. apache.org, last accessed 07-17-2020.

[4] D. F. Aranha, C. P. L. Gouvea, T. Markmann, R. S. Wahby, and ˆ K. Liao. RELIC is an Efficient LIbrary for Cryptography. https: //github.com/relic-toolkit/relic, last accessed 11-25-2020.

[5] ARM Ltd. Mbed OS. https://www.mbed.com, last acc. 07-17-2020.

[6] ARM Ltd. Mbed TLS. https://tls.mbed.org, l. acc. 07-17-2020.

[7] C. Ashokkumar, B. Roy, B. S. V. Mandarapu, and B. Menezes. "SBox" Implementation of AES Is Not Side Channel Resistant. Journal of Hardware and Systems Security, 4:86–97, 2019.

[8] E. Baccelli, C. Gundogan, O. Hahm, P. Kietzmann, M. Lenders, H. Pe- ¨ tersen, K. Schleiser, T. C. Schmidt, and M. Wahlisch. RIOT: an Open ¨ Source Operating System for Low-end Embedded Devices in the IoT. IEEE Internet of Things Journal, 5(6):4428–4440, December 2018.

[9] D. J. Bernstein and T. Lange. Faster Addition and Doubling on Elliptic Curves. In K. Kurosawa, editor, Advances in Cryptology — ASIACRYPT 2007, volume 4833 of Lecture Notes in Computer Science, pages 29–50. Springer, Berlin, Heidelberg, Germany, 2007.

[10] Cifra. A collection of cryptographic primitives targeted at embedded use. https://github.com/ctz/cifra, last acc. 10-11-2020.

[11] F. Conti, R. Schilling, P. D. Schiavone, et al.. An IoT Endpoint System-on-Chip for Secure and Energy-Efficient Near-Sensor Analytics. IEEE Trans. on Circuits and Systems I, 64(9):2481–2494, 2017.

[12] J. Daemen and V. Rijmen. AES Proposal: Rijndael, 1999.

[13] R. de Clercq, L. Uhsadel, A. Van Herrewege, and I. Verbauwhede. Ultra Low-Power Implementation of ECC on the ARM Cortex-M0+. In Proceedings of the 51st Annual Design Automation Conference, DAC '14, pages 1–6, New York, NY, USA, 2014. ACM.

[14] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - A Lightweight and ¨ Flexible Operating System for Tiny Networked Sensors. In IEEE Local Computer Networks (LCN), pages 455–462, 2004. IEEE ComSoc.

[15] A. Durand, P. Gremaud, J. Pasquier, and U. Gerber. Trusted Lightweight Communication for IoT Systems Using Hardware Security. In 9th International Conference on the Internet of Things (IoT '19), pages 1–4, New York, NY, USA, 2019. ACM.

[16] E. Frimpong and A. Michalas. SeCon-NG: Implementing a Lightweight Cryptographic Library Based on ECDH and ECDSA for the Development of Secure and Privacy-Preserving Protocols in Contiki-NG. In 35th Symposium on Applied Computing (SAC '20), pages 767–769, New York, NY, USA, 2020. ACM.

[17] A. H. Gerez, K. Kamaraj, R. Nofal, Y. Liu, and B. Dezfouli. Energy and Processing Demand Analysis of TLS Protocol in Internet of Things Applications. In International Workshop on Signal Processing Systems (SiPS '18), pages 312–317. IEEE, 2018.

[18] C. Gundogan, C. Ams ¨ uss, T. C. Schmidt, and M. W ¨ ahlisch. IoT Con- ¨ tent Object Security with OSCORE and NDN: A First Experimental Comparison. In Proc. of 19th IFIP Networking Conference, pages 19–27, Piscataway, NJ, USA, June 2020. IEEE Press.

[19] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In M. Joye and J.-J. Quisquater, editors, Cryptographic Hardware and Embedded Systems - CHES 2004, vol. 3156 of LNCS, pages 119–132, 2004.

[20] Intel Corporation. TinyCrypt Cryptographic Library. https:// github.com/intel/tinycrypt, last accessed 07-17-2020, 2017.

[21] Ken MacKay. micro-ecc. http://kmackay.ca/micro-ecc/, last accessed 10-11-2020.

[22] P. Kietzmann, T. C. Schmidt, and M. Wahlisch. A Guideline on Pseu- ¨ dorandom Number Generation (PRNG) in the IoT. Technical Report arXiv:2007.11839, Open Archive: arXiv.org, July 2020.

[23] K. H. Kim, J. Choe, S. Y. Kim, N. Kim, and S. Hong. Speeding up Elliptic Curve Scalar Multiplication without Precomputation. IACR Cryptol. ePrint Arch., (Report 2017/669), 2017.