# SwitchCrypt: Navigating Tradeoffs in Stream Cipher Based Full Drive Encryption

## 1. Motivation: The Limitations of Prior Work

Recent work on Full Drive Encryption shows that stream ciphers achieve significantly improved performance over block ciphers while offering stronger security guarantees. However, optimizing for performance often conflicts with other key concerns like energy usage and desired security properties. In this paper we present SwitchCrypt, a software mechanism that navigates the tradeoff space made by balancing competing security and latency requirements via *cipher switching* in space or time. Our key insight in achieving low-overhead switching is to leverage the overwrite-averse behavior of underlying solid-state storage and the per-unit independence of our drive/filesystem layout. This allows us to trade throughput for reduced energy use and/or certain security properties.

As an example: suppose we have an ARM-based ultra-low-voltage netbook provided to us by our employer. As this is an enterprise device, our employer requires that 1) our drive is fully encrypted at all times and 2) our encrypted data is constantly backed up to an offsite system. The industry standard in full drive encryption is AES-XTS, so we initialize our drive with it. Given these requirements, three primary concerns present themselves.

First, it is well known that FDE using AES-XTS adds significant latency and power overhead to I/O operations, especially on mobile and battery-constrained devices [8, 4, 6]. To keep our drive encrypted at all times with AES-XTS means we must accept this hit to performance and battery life. Worse, if our device does not support hardware accelerated AES, performance can be degraded even further; I/O latency can be as high as 3–5x [5]. Hardware accelerated AES is hardly ubiquitous, and the existence of myriad devices that do not support it cannot simply be ignored, hence Google's investment in Adiantum—an FDE solution for "the next billion users" of low-cost devices without support for hardware accelerated AES [3].

Second, AES-XTS is designed to mitigate threats to drive data "at rest," which assumes an attacker cannot access snapshots of our encrypted data nor manipulate our data without those manipulations being immediately obvious to us. With access to multiple snapshots of a drive's AES-XTS-encrypted contents, an attacker can passively glean information about the plaintext over time by contrasting those snapshots, leading to confidentiality violations in some situations [7, 1]. Similarly, an attacker that can manipulate encrypted bits without drawing our attention will corrupt any eventual plaintext, violating data integrity and, in the worst case, influencing the behavior of software. Unfortunately, in real life, data rarely remains "at rest" in these ways. In our example, our employer requires we back up the contents of our drive to some offsite backup service; this service will receive periodic snapshots of the encrypted state of our drive, violating our "at rest" invariant. These backups occur at a layer above the drive controller, meaning any encryption happening at the Flash Translation Layer (FTL) or below is irrelevant.

Third, our system is battery constrained, placing a cap on our energy budget that can change at any moment as we transition from line power to battery power and back. Our system should respond to these changing requirements without violating any other concerns.

To alleviate the performance concern, we can choose a stream cipher like ChaCha20 rather than the AES-XTS block cipher. Using StrongBox, an encryption driver built for ChaCha20-based FDE, we can achieve on average a 1.7x speedup and a commensurate reduction in energy use [5].

When it comes to the security concern, StrongBox solves both the snapshot and integrity problems by 1) never writing data encrypted with the same key to the same location and 2) tracking drive state using a Merkle tree and monotonic counter supported by trusted hardware to prevent rollbacks. This ensures data manipulations cannot occur and guarantees confidentiality even when snapshots are compared regardless of the stream cipher used. Unfortunately, restoring from a backup necessitates a forced rollback of drive state, potentially opening us back up to confidentiality-violating snapshot comparison attacks [5].

To truly address the security concern requires a cipher with an additional security property: *ciphertext randomization*. Without ciphertext randomization, an attacker can map plaintexts to their ciphertext counterparts during snapshot comparison, especially if they can predict what might be written to certain drive regions. However, with ciphertext randomization, a cipher will output a different "random" ciphertext even when given the same key, nonce, and plaintext; this means, even after a forced rollback of system state and/or legitimate restoration from a backup, comparing future writes is no longer violates confidentiality because each snapshot will always consist of different ciphertext regardless of the plaintext being encrypted or the state of the drive. Using Freestyle [2], a ChaCha20-based stream cipher that supports ciphertext randomization, we can guarantee data confidentiality in this way. So, we switch from StrongBox to an encryption driver that supports Freestyle.

Unfortunately, like AES-XTS, Freestyle has significant overhead compared to the original ChaCha20. In exchange for stronger security properties, Freestyle is up to 1.6x slower than ChaCha20, uses more energy, has a higher initialization

cost, and expands the ciphertext which reduces total writeable drive space [2].

Further complicating matters is our final concern: a constrained energy budget. Our example system is battery constrained. Even if we accepted trading off performance, drive space, and energy for security in some situations, in other situations we might prioritize reducing total energy use. For example, when we trigger "battery saver" mode, we expect our device to conserve as much energy as possible. It would be ideal if our device could pause backups and the encryption driver could switch from the ciphertext-randomizing Freestyle configuration back to our high performance energy-efficient ChaCha20 configuration when conserving energy is a top priority, and then switch back to the Freestyle configuration when we connect to a charger and backups are eventually resumed.

Hence we present SwitchCrypt, a device mapper that can trade off between these two configurations and others without compromising security or performance or requiring the device be restarted. With prior work, the user must select a static operating point in the energy-security-latency space at initialization time and hope it is optimal across all workloads and cases. If they choose the Freestyle configuration, their device will be slower and battery hungry even when they are not backing up. If they choose the more performant ChaCha20 configuration, they risk confidentiality-violating snapshot comparison attacks. SwitchCrypt solves this problem by encrypting data with the high performance ChaCha20 when the battery is low and switching to Freestyle when plugged in and syncing with the backup service resumes.

## 2. Challenges

To trade off between different cipher configurations, we must address three key challenges. First, we must determine what cipher configurations are most desirable in which contexts and why. This requires we *quantify* the desirable properties of these configurations. Second, we must have some way to encrypt independent storage units with any one of these configurations. This requires we *decouple* cipher implementations from the encryption process used in prior work. Third, we need to determine when to re-encrypt those units, which configuration to use, and where to store the output, all with minimal overhead. This requires we implement efficient cipher *switching strategies*.

**Quantifying the properties traded off between configurations.** To obtain a space of configurations that we might reason about, it is necessary to compare certain properties of stream ciphers useful in the FDE context. However, different ciphers have a wide range of security properties, performance profiles, and output characteristics, including those that randomize their outputs and those with non-length-preserving outputs—*i.e.,* the cipher outputs more data than it takes in. To address this, we propose a framework for quantitative cipher comparison in the FDE context; we use this framework to define our configurations.

**Decoupling ciphers from the encryption process.** To flexibly switch between configurations in SwitchCrypt requires a generic cipher interface. This is challenging given the variety of inputs required by various stream ciphers, the existence of non-length-preserving ciphers, and other differences. We achieve the required generality by defining independent storage units called *nuggets*; we borrow this terminology from prior work (see [5]) to easily differentiate our logical blocks (nuggets) from physical drive and other storage blocks. And since they are independent, we can use our interface to select any configuration to encrypt or decrypt any nugget at any point.

**Implementing efficient switching strategies.** Finally, to determine when to switch a nugget's cipher and to where we commit the output, we implement a series of high-level policies we call *cipher switching strategies*. These strategies leverage our generic cipher interface and flexible drive layout to selectively "re-cipher" groups of nuggets, whereby the key and the cipher used to encrypt/decrypt a nugget are switched at runtime. These strategies allow SwitchCrypt to move from one configuration point to another or even settle on optimal configurations wholly unachievable with prior work. The challenge here is to accomplish this while minimizing overhead.

## 3. Contributions and Results

- Define a scheme to *quantify* the usefulness of each cipher based on key security properties relevant to FDE in context. Using this scheme, we define a tradeoff space of cipher configurations over competing concerns: total energy use, desirable security properties, read and write performance (latency), total writable space on the drive, and how quickly the contents of the drive can converge to a single encryption configuration.
- Introduce a novel design substantively expanding prior FDE work by wholly *decoupling* cipher implementations from the encryption process.
- Develop the idea of cipher switching using *switching strategies* to "re-cipher" storage units dynamically, allowing us to tradeoff different performance and security properties of various configurations at runtime.

We implement SwitchCrypt and three switching strategies—*Forward*, *Selective*, and *Mirrored*—to dynamically transition the system between configurations using our generic cipher interface. We then study the utility of cipher switching through three case studies where latency, energy, and desired security properties change over time. In one study, where we require the filesystem to react to a shrinking energy budget, we find that SwitchCrypt achieves up to a 3.3x total energy use reduction compared to a static approach using only the Freestyle stream cipher. In another case, where we allow the user to manually switch between ChaCha20 and Freestyle stream ciphers dynamically, we achieve a 3.1x reduction in read latency compared to static approaches.

## References


[1] The xts-aes tweakable block cipher, 2008. IEEE Std 1619-2007.

[2] P. Arun Babu and Jithin Jose Thomas. Freestyle, a randomized version of chacha for resisting offline brute-force and dictionary attacks. Cryptology ePrint Archive, Report 2018/1127, 2018. https://eprint.iacr.org/2018/1127.

[3] Paul Crowley and Eric Biggers. Adiantum: length-preserving encryption for entry-level processors. *IACR Transactions on Symmetric Cryptology*, 2018(4):39–61, 2018.

[4] Andrew Cunningham and Utc. Google quietly backs away from encrypting new lollipop devices by default, Mar 2015.

[5] Bernard Dickens III, Haryadi S. Gunawi, Ariel J. Feldman, and Henry Hoffmann. Strongbox: Confidentiality, integrity, and performance using stream ciphers for full drive encryption. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 708–721, New York, NY, USA, 2018. ACM.

[6] Joshua Ho and Brandon Chester. Encryption and storage performance in android 5.0 lollipop, Nov 2014.

[7] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes ocb and pmac. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, pages 16–31, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[8] Timothy J. Seppala. Google won't force android encryption by default, Jul 2019.