On the Practical (In-)Security of 64-bit Block Ciphers

Collision Attacks on HTTP over TLS and OpenVPN

Karthikeyan Bhargavan
Inria, France
karthikeyan.bhargavan@inria.fr

Gaëtan Leurent
Inria, France
gaetan.leurent@inria.fr

ABSTRACT

While modern block ciphers, such as AES, have a block size of at least 128 bits, there are many 64-bit block ciphers, such as 3DES and Blowfish, that are still widely supported in Internet security protocols such as TLS, SSH, and IPsec. When used in CBC mode, these ciphers are known to be susceptible to collision attacks when they are used to encrypt around 2^{32} blocks of data (the so-called birthday bound). This threat has traditionally been dismissed as impractical since it requires some prior knowledge of the plaintext and even then, it only leaks a few secret bits per gigabyte. Indeed, practical collision attacks have never been demonstrated against any mainstream security protocol, leading to the continued use of 64-bit ciphers on the Internet.

In this work, we demonstrate two concrete attacks that exploit collisions on short block ciphers. First, we present an attack on the use of 3DES in HTTPS that can be used to recover a secret session cookie. Second, we show how a similar attack on Blowfish can be used to recover HTTP BasicAuth credentials sent over OpenVPN connections. In our proof-of-concept demos, the attacker needs to capture about 785GB of data, which takes between 19-38 hours in our setting. This complexity is comparable to the recent RC4 attacks on TLS: the only fully implemented attack takes 75 hours. We evaluate the impact of our attacks by measuring the use of 64-bit block ciphers in real-world protocols. We discuss mitigations, such as disabling all 64-bit block ciphers, and report on the response of various software vendors to our responsible disclosure of these attacks.

1. INTRODUCTION

Internet protocols such as TLS [13], SSH [34], and IPsec [18] are *agile*, in the sense that they are designed to support a wide variety of *ciphersuites*: combinations of key exchange protocols, encryption schemes, authentication modes, etc. Each protocol implementation may choose to support a different subset of ciphersuites, but two implementations can still interoperate if they can negotiate a common ciphersuite. When it works well, protocol agility can enable a graceful transition from old cryptographic algorithms to new ones. A server can, for example, offer AES-GCM to modern clients while still supporting legacy ciphers like 3DES for older clients that have not yet been upgraded. However, a negative consequence of agility is that old ciphers may never be removed, resulting in implementations that support a few strong modern ciphers, followed by a long tail of obsolete ciphers that are still supported for backwards compatibility, but are known to be cryptographically weak. For example, the OpenSSL library supports five versions of TLS and hundreds of ciphersuites, even though many of these ciphersuites include weak algorithms like RC4 and MD5.

There are several reasons why practitioners do not consider theoretical weaknesses in cryptographic to be sufficient for their removal from mainstream protocols. First, even if an obsolete primitive is supported, it will typically only be negotiated if one of the parties does not support a modern alternative, in which case, the obsolete cipher is still better than nothing. Second, the attack may not be applicable to the way the primitive is used in the protocol, or it may require too much computation to be considered practical. Third, the attack may require special knowledge about the structure or the content of the plaintext stream which may be difficult to obtain in general. Consequently, protocol implementations tend to support legacy ciphers until a concrete attack is demonstrated in a common usage of the protocol.

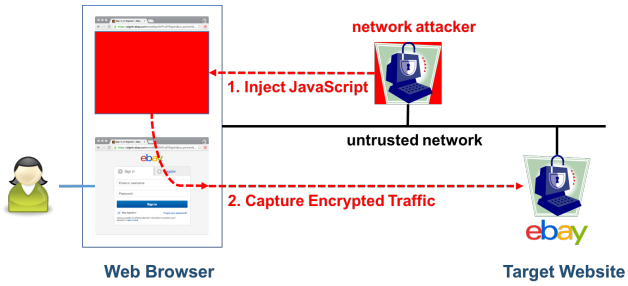

Figure 1: The Beastly attacker model for HTTPS: A network attacker injects JavaScript on some window in the user's browser, which then makes cross-origin HTTPS requests to a target website. The browser attaches a secret cookie or HTTP BasicAuth password to authenticate each request. The attacker observes the encrypted traffic stream and tries to recover the secret.

CCS '16, October 24–28, 2016, Vienna, Austria.

ACM ISBN 978-1-4503-4139-4/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978423

Attacks on Legacy Crypto. A series of recent attacks on TLS have conclusively shown that this line of defense for legacy ciphers is fatally flawed. Leaving old protocol versions and obsolete cryptographic algorithms enabled can lead to devastating attacks, such as BEAST [15] (on CBC encryption in TLS 1.0), CRIME [28] (on TLS compression), Lucky13 [4] (on CBC padding in TLS 1.2), POODLE [11] (on CBC padding in SSL 3.0), RC4 NOMORE [3, 19, 32] (on keystream biases in RC4), FREAK [9] (on export-grade RSA keys), Logjam [2] (on export-grade Diffie-Hellman groups), SLOTH [10] (on MD5 signatures), and DROWN [5] (on RSA-PKCS#1v1.5 encryption in SSL 2.0).

These attacks follow a common pattern. Each attack is based on a classic cryptographic weakness that was known to cryptographers for years or even decades in advance, but had not been demonstrated in real-world TLS scenarios. So these works showed that with modern network speeds and computing power, the computational effort required to exploit the theoretical weakness is feasible even for academic researchers. Furthermore, many of these attacks (e.g. BEAST, CRIME, RC4 NOMORE) can be made more efficient by relying on a new HTTPS attacker model introduced by BEAST. Figure 1 depicts this Beastly attacker model, where a network attacker uses JavaScript to implement an adaptive chosen-plaintext adversary who can mix in his plaintext with the user's secret values, and then use his network vantage point to actively or passively attack the ciphertext.

The final element of each attack is finding clients and servers who are willing to negotiate the legacy cipher. Some of the attacked algorithms (e.g. RC4) and protocol versions (e.g. SSL 3.0) were highly popular before the attack and have since been deprecated. Other weak algorithms (e.g. export ciphers) and versions (e.g. SSL 2.0) had already been deprecated years ago and were never expected to be negotiated between modern clients and servers. Surprisingly however, in many cases the attacker is able to force the negotiation of a weak algorithm via a *downgrade attack* by exploiting protocol flaws (e.g. Logjam, SLOTH) or implementation bugs (e.g. FREAK, DROWN). One lesson to take from these downgrade attacks is that relying on the negotiation protocol to avoid legacy ciphers is error-prone and dangerous. When a concrete attack against a legacy cipher is discovered, the only safe mitigation is to fully remove the weak cipher from all implementations.

These previous attacks evaluated a variety of obsolete cryptographic constructions and provided enough evidence of their dangers for them to be disabled in popular implementations. We believe that it is important for the research community to be vigilant and to continue to investigate the strength of the weakest ciphers still supported by mainstream protocols. For example, the attacks on RC4 have resulted in RC4 being removed as the default fallback cipher for TLS implementations, and this role has now fallen to 3DES, which is a 28 year old 64-bit block cipher with known weaknesses. However, the practical security of 64-bit block ciphers as used in TLS and other protocols has not been previously studied. This work aims to address this gap and to provide concrete attacks and firm guidance on the use of such ciphers.

Collision Attacks on 64-bit Block Ciphers. The security of a block cipher is often reduced to the key size k: the best attack should be the exhaustive search of the key, with complexity 2^k. However, the block size n is also an important security parameter, defining the amount of data that can be encrypted under the same key. This is particularly important when using common modes of operation: we require block ciphers to be secure with up to 2^n queries, but most modes of operation (*e.g.* CBC, CTR, GCM, OCB, etc.) are unsafe with more than $2^{n/2}$ blocks of message (the so-called birthday bound). In particular, when these many blocks are encrypted under a key in CBC mode, the probability of collisions between two ciphertext blocks becomes significant, and each ciphertext collision reveals the XOR of the two corresponding plaintext blocks from the stream. Consequently, if the attacker can guess one of the plaintext blocks, he can recover the plaintext in the other block. While this attack is well-known to researchers [30, 25], it is often disregarded by practitioners because it require known plaintext, and reveals only a few bits of a large datastream.

With a modern block cipher with 128-bit blocks such as AES, the birthday bound corresponds to 256 EB (2^{68} bytes). While this is leaves a safe margin for common uses of cryptography today, the resulting security is still lower than could be expected of 128-bit security and might be an issue in the future. As a point of comparison, Google is estimated to store around 16 EB (2^{64} bytes), and the global Internet traffic is expected to reach 1 ZB (2^{70} bytes) per year in 2016.

More strikingly, the birthday bound corresponds to only 32 GB for a block cipher with 64-bit blocks, and there are many real-world protocol scenarios in which this amount of data may be sent under a 64-bit cipher: e.g. 3G telephone connections (UMTS), which are encrypted with KASUMI, OpenVPN tunnels, which uses Blowfish as the default cipher, and Internet standards like TLS, SSH, and IPsec that use 3DES and Blowfish as legacy ciphers for interoperability. In all these scenarios, 32 GB of data can be transferred in less than one hour with a fast connection.

More recently, 64-bit ciphers have become common in the field of *lightweight cryptography* for applications, such as embedded systems and smart devices, where resources are too constrained to use conventional block ciphers. For instance, PRESENT [12] and HIGHT [20] have been standardized by ISO recently (in ISO/IEC 29192 and ISO/IEC 18033-3, respectively).

Our Results. In this work, we study the concrete impact of birthday attacks against 64-bit block-ciphers as used in popular protocols. We present the first concrete attacks based on ciphertext collisions on mainstream Internet protocols.

We focus on two concrete use cases. First, we consider the use of OpenVPN to secure HTTP connections between a web browser and a website, where the website asks the user for HTTP BasicAuth (user/password) authentication. OpenVPN uses Blowfish in CBC mode by default, and we show how one can use the Beastly attacker scenario to generate a lot of traffic including the BasicAuth in every request, until collisions collisions in the Blowfish ciphertext reveal the BasicAuth token (*i.e.* the password). We demonstrate a proof-of-concept attack between a Firefox client and an nginx server connected via OpenVPN. The attack requires about 785GB, which takes 19 hours in our setting to recover the 16-byte authorization token.

Second, we consider the use of TLS to secure HTTP connections between a browser and an HTTPS website that sets secure session cookies on the browser. We find that 3DES is supported by nearly 90% of HTTPS servers and by all mainstream TLS clients. In practice, however, 3DES is less preferred than AES and negotiated by only 1-2% of

HTTPS connections. For such connections, we demonstrate an attack, again using the Beastly attacker setup, that can recover the secret 16-byte session cookie sent by the browser with every HTTPS request to the target server. We demonstrate the attack between a Firefox browser and an IIS 6.0 server (Windows Server 2003 R2 SP2), both in their default up-to-date configuration. This attack also requires 785GB, which takes 38 hours in our setting.

Our attacks demonstrate that collision-based attacks on 64-bit block ciphers are practically exploitable. We suggest two mitigations for the attack. Considering the low negotiation probability of 3DES in TLS, we recommend disabling it in popular clients and servers. For protocols where 64-bit ciphers cannot yet be deprecated, we recommend using strict rekeying limits to mitigate the attack. In response to our attacks, mainstream TLS libraries, web browsers, and OpenVPN have begun implementing these mitigations.

Outline. Section 2 provides background on collision attacks on block ciphers and estimates their success probability under various conditions. Section 3 investigates the usage of 64-bit ciphers in various Internet protocols, and specifically in HTTPS. Section 4 shows how block cipher collisions in the encrypted stream can be used to attack HTTP. Section 4.4 and 4.5 describe our proof-of-concept attacks on OpenVPN and HTTPS, respectively. Section 5 discusses the impact of our attacks and their mitigations.

2. COLLISION ATTACKS ON BLOCK CIPHERS

Block ciphers like AES and 3DES are widely used for symmetric encryption in security protocols. Mathematically, a block cipher with a κ-bit key and n-bit blocks defines of family of permutations of n-bit strings, indexed by the key. The main security notion is that a block cipher should behave like a pseudo-random permutation (PRP) family: the block cipher E_k with a random key k should be hard to distinguish from a random permutation.

Block ciphers operate on blocks of fixed size, n bits, usually with $n = 64$ (DES, 3DES, Blowfish, IDEA, KASUMI, PRESENT, ...) or $n = 128$ (AES, Camellia, Twofish, Serpent, SEED, ...). In order to deal with arbitrary-length messages, they are used within a *mode* of operation. A mode divides the message M into n-bit blocks m_i, and process the blocks one by one through the block cipher E with various chaining rules, to produce ciphertext blocks c_i and/or an authentication tag τ. Several modes have been widely studied and standardized [16] for encryption (CBC, CFB, OFB, CTR), authentication (CBC-MAC, PMAC) and authenticated encryption (OCB, CCM, GCM).

2.1 Security at the Birthday Bound

While block ciphers are required to resist any attack with up to 2^n data and 2^κ time, most modes of operation are only proven secure up to $2^{n/2}$ blocks of plaintext, a limit that is commonly called the *birthday bound*. Indeed, there are attacks matching this limit. In CBC mode, the probability of collisions between two n-bit ciphertext blocks becomes significant after $2^{n/2}$ blocks because of the birthday paradox. Rogaway summarizes the situation as follows [30, page 36]:

> In general, unless special efforts are taken, almost any mode of operation based on an n-bit block-cipher will admit attacks that employ $\sigma = 2^{n/2}$ blocks. [...]
>
> Do birthday-bound attacks on CBC, CFB, and OFB actually matter? They are of relatively little concern when the blockcipher has a blocksize of $n = 128$ bits, but the attacks can be a serious concern when employing a blockcipher of $n = 64$ bits, requiring relatively frequent rekeying to keep $\sigma \ll 2^{32}$.

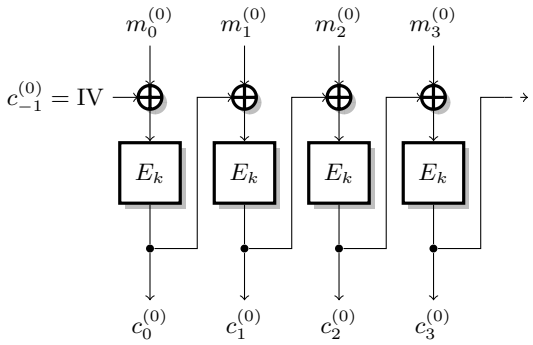

Figure 2: CBC mode

More recently, McGrew investigated plaintext recovery attacks on 64-bit ciphers in CBC, CFB, and CTR modes. He studied the mount of information leakage by assuming that the attacker can observe all the ciphertext in an encrypted stream and he has some (incomplete) knowledge of the plaintext [25]. This theoretical analysis resulted in new security recommendations for IPsec [26], but they did not trigger any significant changes in mainstream deployments of 64-bit ciphers in IPsec, TLS, or SSH. Arguably the main reason for the limited impact of these attacks is that they were never implemented against a popular protocol under real-world conditions.

Following McGrew, we explain theoretical attacks against the CBC and CTR modes (the two most widely used encryption modes) in more detail below. In the rest of the paper, we will show that these attacks can be exploited to mount concrete attacks on HTTP over TLS and OpenVPN.

2.2 Plaintext recovery attack against CBC

The CBC mode is one of the oldest encryption modes, and still widely used. The message M is divided into blocks $m_i, 0 \le i < \ell$ (see Figure 2) and is encrypted as:

$$c_i = E_k(m_i \oplus c_{i-1}),$$

where c_{-1} is an initialization value usually denoted as IV. For simplicity, here we assume that all message sizes are a multiple of the block size.

When encrypting a sequence of messages $M^{(j)}$, each message should be encrypted with a fresh random IV $c_{-1}^{(j)}$. It is important that the IV can not be predicted by an adversary who can encrypt chosen messages; otherwise, it can lead to attacks, as first observed by Rogaway in 1995 [29], and subsequently exploited in the BEAST attack [15] in 2011 on TLS versions prior to 1.1.

CBC has been proven secure up to $2^{n/2}$ blocks of messages [7, 27]. On the other hand there is a simple birthday attack against CBC: after $2^{n/2}$ message blocks encrypted

with the same key (in the same message or in different messages), a collision between two ciphertext blocks $c_i^{(j)} = c_{i'}^{(j')}$ is expected. Since E_k is a permutation, a collision in the output means that the inputs are the same ($m_i^{(j)} \oplus c_{i-1}^{(j)} = m_{i'}^{(j')} \oplus c_{i'-1}^{(j')}$), which reveals the xor of two plaintext blocks:

$$m_i^{(j)} \oplus m_{i'}^{(j')} = c_{i-1}^{(j)} \oplus c_{i'-1}^{(j')}.$$

Success probability. In general, with 2^d blocks of data, the probability that there is at least one collision is (using $D = 2^d$ and $N = 2^n$):

$$\begin{aligned} p &= 1 - \prod_{i=0}^{D-1} \frac{N-i}{N} \\ &\geq 1 - \prod_{i=0}^{D-1} e^{-i/N} && \text{using } 1 - x \leq e^{-x} \\ &\geq 1 - e^{-D(D-1)/2N} \end{aligned}$$

In particular, with $d > n/2$, we have $p \geq 1 - e^{-1/2} \approx 0.39$. With larger values of d, the probably grows very close to 1:

$$\begin{aligned} p &\geq 1 - \frac{2N}{D(D-1)+2N} && \text{using } 1 - e^{-x} > \frac{x}{x+1} \\ &\geq 1 - \frac{2N}{D^2} && \text{since } D^2 < D(D-1) + 2N \\ p &\geq 1 - 2^{n-2d+1} \end{aligned}$$

On the other hand, when $d < n/2$ we can use:

$$p \geq \frac{D(D-1)}{4N} \approx 2^{2d-n-2} \qquad \text{using } e^{-x} < 1 - x/2$$

Furthermore, with 2^d blocks of data the expected number of collisions is roughly 2^{2d-n-1}.

2.3 Distinguishing attack against CTR

The counter mode CTR is another popular mode of operation for encryption, first proposed by Diffie and Hellman [14]. In particular, it provides the encryption part of authenticated encryption modes CCM and GCM, and a variant of counter mode is used in 3G telephony (f8). The counter mode actually builds a stream cipher out of a block cipher: a counter is encrypted to generate key-stream that is xored with the message. More precisely, the message M is divided into blocks $m_i, 0 \leq i < \ell$ which are encrypted as:

$$c_i = E_k(\text{IV} + i) \oplus m_i,$$

where IV is the initial value of the counter (see Figure 3).

If the encryption function E_k is a pseudo-random *function*, rather than a pseudo-random *permutation*, then CTR is secure up to 2^n blocks [6]. However, in the common case where E_k is a permutation, CTR is proven secure only up to $2^{n/2}$ blocks, and there is matching distinguishing attack with complexity $2^{n/2}$. If a message where all the plaintext blocks are equal is encrypted, then all the ciphertext blocks will be pairwise different. Indeed, all the $E_k(\text{IV} + i)$ are unique, because E_k is a permutation. This enables the attacker to distinguish the counter mode from an ideal encryption scheme, because a collision between two ciphertext blocks becomes likely (with probability about 0.39). More interestingly, if the attacker has prior knowledge about the plaintext (such as a dictionary), he can use the distinguishing attack to eliminate *impossible values* to eventually guess the right plaintext [25].

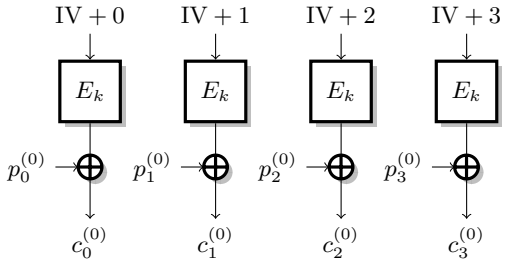

Figure 3: CTR mode

It is interesting to compare the collision attacks against CBC and CTR. Both attacks have the same complexity, and show that the proofs are tight. However, the loss of security is quite different: an attacker can immediately recover some bits of information from the attack against CBC, while the attack against the counter mode requires a lot more work to recover any plaintext. In the remainder of this paper, we investigate on the practical impact of the CBC attack, but similar techniques (with higher complexity and more assumptions on the plaintext) may be used to attack CTR and GCM mode encryption in protocols like TLS and SSH.

2.4 Rekeying before the Birthday Bound

The main countermeasure against birthday attacks is to change the key frequently, before reaching the birthday bound under any single key. For instance, if the key is changed after $2^{n/2-16}$ blocks, a birthday attack against a given only succeeds with probability 2^{-32}, and an attacker has to collect 2^{32} different sessions to mount an attack with high probability.

If implemented properly, this is an effective security measure to strengthen block ciphers with a small block-size. We stress than the key must be changed *well before* $2^{n/2}$ blocks, despite misleading recommendations in published standards. For instance, ISO recommends [21]:

> Generally, for a block cipher with block size of n bits, the maximum amount of plaintext that can be encrypted before rekeying must take place is $2^{(n/2)}$ blocks, due to the birthday paradox. As long as the implementation of a specific block cipher do not exceed these limits, using the block cipher will be safe.

This is *incorrect*, because collisions can be found with high probability (close to 0.39) with only $2^{n/2}$ blocks. Indeed, we will show that our concrete attacks also work against connections that are rekeyed close to the birthday bound.

2.5 Towards a Practical Attack on CBC

The CBC attack reveals the xor of two plaintext block, and in many contexts, this may not be sufficient for a practical attack. However, an effective attack can be mounted when the following conditions are fulfilled: (1) some high-value secret data is sent repeatedly; and (2) some fraction of the plaintext is known. In this case, there is a chance that a collision leaks the xor between a valuable secret and a known block; this would immediately reveal the secret. More precisely, this attack is expected to succeed (*i.e.* to recover one block of a high-value secret) with high probability as

soon as 2^s blocks of secret and 2^t known blocks are encrypted, with $s + t \geq n$.

To mount the attack in practice, an adversary would collect and store all the traffic, then sort the ciphertext blocks to find collisions. Assuming that the adversary knows the position of the secret blocks and of the known blocks in the stream, he can quickly see which collisions are useful, and recover the secret values by xoring with the known plaintext.

Attack Complexity. Let us denote the known fraction of the data as α and the secret and valuable fraction of the traffic as β (with fraction $1 - \alpha - \beta$ that is neither known nor valuable to the attacker). In order to recover some secret information, an attacker must collect roughly $1/2\alpha\beta$ collisions, so that one collision is between a valuable secret block and a known block. Following the analysis of Section 2.2, this requires about $\sqrt{1/\alpha\beta} \cdot 2^{n/2}$ blocks of data.

For instance, with $n = 64$, in an optimal case for the adversary half of the traffic is known, and half of the traffic is highly valuable ($\alpha = \beta = 1/2$). In this case, a collision an attack requires about:

$$\sqrt{1/\alpha\beta} \cdot 2^{n/2} = 2 \cdot 2^{n/2} = 2^{33}$$

blocks of data, which corresponds to just 64 GB. For a more concrete scenario, let us assume that the messages are HTTP queries of 512 bytes (64 blocs), with a secret 8-byte cookie (1 block), and that the remaining of the message is known, *i.e.* $\alpha = 63/64 \approx 1$, $\beta = 1/64$. The number of blocks needed by an attacker is roughly:

$$\sqrt{1/\alpha\beta} \cdot 2^{n/2} \approx 8 \cdot 2^{n/2} = 2^{35}$$

which correspond to 256 GB.

Rekeying at the Birthday Bound. The attack can also be carried out when rekeying is used, if the keys are changed with low frequency. In particular, even if the keys are changed before $2^{n/2}$ blocks are encrypted, there is still a non-negligible probability that a collision occurs.

More precisely, let us assume that keys are rotated after encrypting 2^r blocks ($r \leq n/2$). As explained in Section 2.2, the probability to have a collision between 2^r blocks is at least 2^{2r-n-2}. Therefore, an attacker needs $1/2\alpha\beta \cdot 2^{n+2-2r}$ sessions (*i.e.* $1/2\alpha\beta \cdot 2^{n+2-r}$ blocks) to collect an interesting collision. In practice, the attacker would collect all the data in a session, sort it to find collisions, and can discard the data before collecting data from the next session.

In an optimal setting for the attacker ($\alpha = \beta = 1/2$), if the keys are changed after $2^{n/2}$ blocks as recommended by ISO [21] ($r = n/2$), the number of blocks needed by an attacker is roughly:

$$1/2\alpha\beta \cdot 2^{n+2-r} = 2 \cdot 2^{n/2+2} = 2^{35}$$

which correspond to 256 TB. In the HTTP example above, the number of blocks needed is roughly:

$$1/2\alpha\beta \cdot 2^{n+2-r} \approx 32 \cdot 2^{n/2+2} = 2^{39}$$

which correspond to 4 TB. Therefore rekeying just before $2^{n/2}$ blocks only has a limited impact on collision attacks.

Optimal Rekeying Strategies. The optimal use of rekeying has been studied by Abdalla and Bellare in [1], with the goal of increasing the *encryption threshold* (the number of blocks that can be encrypted before the advantage of an adversary becomes significant). Their result depend on the key size κ and the block size n. When $n = \kappa$, the optimal strategy is to rekey after $2^{n/3}$ blocks, leading to an encryption threshold of $2^{2n/3}$ (rather than $2^{n/2}$ without rekeying). Concrete figures for AES-based modes are also given in [24].

On the other hand, when $\kappa \geq 2n$ (as is the case for 3DES and Blowfish-128), the optimal strategy is to rekey for every block. Assuming a perfect rekeying function, this leads to an encryption threshold of 2^n (rather than $2^{n/2}$). In actual protocols such as TLS or OpenVPN, rekeying requires a new key exchange, which can be quite expensive. Therefore, rekeying should probably be implemented with a fixed limit, in the order of a few megabytes. We can compute the *encryption threshold* when rekeying every 2^r blocks as 2^{n-r} (this figure correspond to the optimal case for an adversary, with $\alpha = \beta = 1/2$). For example, by rekeying every megabyte, a client and server can exchange a petabyte of data before collision attacks become significant.

Exploiting Repeated Messages. We now assume that a fixed message of 2^u blocks is sent repeatedly (this will be the case for the concrete attacks we describe in Sections 4.4 and 4.5). The attacker's goal is to recover the full message, assuming that $\alpha \cdot 2^u$ blocks are known or predictable.

If there is a single unknown block, we can use the previous analysis with $\alpha \approx 1$ and $\beta = 2^{-u}$: an attacker need roughly $2^{(n+u)/2}$ blocks. Indeed, this should lead to roughly 2^{u-1} collisions, and there is a good chance that one of these collision is between the secret block and a known block. For a more accurate analysis, we can use the results of van Oorschot and Wiener [31], which gives an expected complexity of $\sqrt{\pi/2} \cdot 2^{(n+u-1)/2}$ blocks to find 2^{u-1} collisions.

If there are a few unknown blocks ($k = (1 - \alpha) \cdot 2^u$), the number of encrypted blocks required to recover them can be approximated as:

$$\sqrt{H_k} \cdot \sqrt{\pi/2} \cdot 2^{(n+u-1)/2},$$

because the number of collision required follows the harmonic numbers [1] $H_k = \sum_{i=1}^{k} 1/i$.

On the other hand, if α is small, the attacker can use collisions to derive equations between the plaintext blocks, even if both blocks are unknown. Each collision reveals the xor between two plaintext blocks, which can seen as an edge in a graph whose vertexes are the plaintext blocks. If we model the graph as a random graph with 2^u vertexes, we know that with high probability, the graph will be fully connected when there are more than $\Theta(u/2 \cdot 2^u)$ edges [17]. At this point, all plaintext blocks can be recovered from a single known block. Therefore, the expected attack complexity becomes:

$$\sqrt{u} \cdot \sqrt{\pi/2} \cdot 2^{(n+u-1)/2}$$

Summary of Attack Scenario. With a fixed message of size 2^u repeatedly encrypted, an attacker that knows a fraction α of the message can recover the k missing blocks of plaintext ($k = (1 - \alpha) \cdot 2^u$) by observing about $\log(k) \cdot 2^{u-1}$ collisions. This gives an attack with a data complexity of

$$O\big(\sqrt{\log(k)} \cdot 2^{(n+u-1)/2}\big)$$

encrypted blocks. With rekeying every 2^r blocks ($r < n/2$), the complexity becomes:

$$O\big(\log(k) \cdot 2^{u+n+2-r}\big).$$

[1] Assuming that the time to recover each block follows a geometric law

Protocol	Year	Block ciphers	Mandatory	Rekey
TLS 1.0	1999	**3DES, DES, IDEA**	**3DES**	-
TLS 1.1	2006	AES, **3DES, DES**	**3DES**	2^{78}
TLS 1.2	2008	AES, **3DES**	AES	2^{78}
SSH 1	1995	**3DES, DES, IDEA**	**3DES**	-
SSH 2	2006	AES, **3DES, Blowfish**	**3DES**	2^{30},
IKEv1	1998	**3DES, DES, Blowfish**	**DES**	-
IKEv2	2010	AES, **3DES, Blowfish**	**3DES**	-
IPsec	2014	AES, **3DES**	AES	1GB

Table 1: Block ciphers in popular versions of Internet protocols; 64-bit ciphers are shown in red

In particular, we will apply this attack to break the security of encrypted HTTP connections when an authentication token is send with every request. We use the Beastly attacker of Figure 1 to generate a large number of requests, where the headers are mostly predictable or even controlled by the attacker ($\alpha \approx 1$), who can then recover the authentication token by observing collisions in the encrypted traffic.

3. 64-BIT BLOCK CIPHER USAGE ON THE INTERNET

Many of the most influential Internet security protocols, such as TLS, SSH, and IPsec were standardized at a time when 64-bit block ciphers, such as 3DES and Blowfish, were still considered strong. Consequently, these ciphers are still widely supported on the Internet. In this section, we investigate the real-world usage of these ciphers in order to quantify the feasibility and practical impact of our attacks.

3.1 Block Ciphers in Internet Standards

Table 1 summarizes the support for various block ciphers in several versions of the TLS, SSH, and IPsec standards. The table shows that protocols that were standardized before 2000 typically supported only 64-bit ciphers (primarily 3DES), although later protocol extensions enabled 128-bit ciphers (primarily AES). Still, old versions of these protocols remain quite popular on the web. Notably, each standard specifies a *mandatory* cipher that "MUST" be supported for interoperability, and in most of these specifications, the mandatory encryption algorithm is 3DES. This means that all implementations of TLS, SSH, and IPsec must implement 3DES, and unless the user explicitly disables it, they will offer to negotiate 3DES with their peers.

For example, in TLS 1.0 and 1.1, 3DES is the mandatory encryption algorithm, so all TLS libraries implement it and a vast majority of web servers support it. Although TLS 1.2 changed the mandatory cipher to AES, it still explicitly recommends 3DES as secure [13, Sec. 7.3]:

> If you negotiate 3DES with a 1024-bit RSA key exchange with a host whose certificate you have verified, you can expect to be that secure.

As a result, mainstream TLS libraries still treat 3DES to be at the same security level as AES-128. For example, until we disclosed the attacks in this paper, OpenSSL included 3DES ciphersuites in its HIGH-security list (it has now been moved to MEDIUM). Moreover, while the use of RC4 in TLS has been explicitly deprecated in RFC7465 in response to recent attacks, 3DES continues to be trusted by TLS libraries. In fact, it has even started being recommended as the new backup cipher in place of RC4.[2] TLS does not mandate rekeying for short block ciphers, and so TLS libraries leave the decision to rekey to their applications.

In SSH version 1, the strongest available ciphers are all 64-bit block ciphers: 3DES, Blowfish, and IDEA, and although SSH 2 introduced AES-based ciphers, it still labeled 3DES as the mandatory cipher. Consequently, popular implementations such as OpenSSH still use 3DES as the default cipher for SSH-1 connections, and 3DES continues to be the most widely supports cipher in SSH.[3] SSH standards recommend that implementations rekey after every 1GB of data, or after every $2^{n/4}$ blocks [8], but many popular SSH clients do not implement this feature.

The early IPSec specifications also widely used and recommended 3DES. For example, RFC4835 recommended 3DES as a mandatory (MUST) algorithm until 2007, although the more recent RFC7321 has downgraded it to a "MAY". Consequently, most IPSec libraries continue to support 3DES, and since IPSec deployments tend to be stable over a long period of time within corporate networks, we anticipate that the algorithm may continue to be supported on IPSec networks for the foreseeable future.

In summary, short block ciphers and 3DES in particular have widespread support in major secure transport protocols, making them vulnerable to the attacks described in this paper. However, to evaluate the real-world impact of this vulnerability, the key question is whether these ciphers are actually used in connections between popular clients and servers. In the rest of this section, we attempt to quantify this usage.

3.2 3DES and Blowfish usage in VPNs

TLS, SSH, and IPSec are often used to implement *tunnels* or virtual private networks (VPNs) across untrusted network connections. We have already discussed the widespread support for 64-bit block ciphers in these protocols, but estimating whether these ciphers are actually negotiated in practice is hard, because unlike web servers, most VPN services are not publicly accessible, and each VPN server may use its own configuration. However, we found a few VPN solutions that use 64-bit ciphers by default.

IPsec. Most IPSec-based VPN clients support 3DES for interoperability. Notably, some versions of Microsoft's L2TP VPN client use 3DES by default[4]; some versions of MAC OS X negotiate 3DES with certain VPN servers[5]; and some versions of CISCO IPsec VPN devices implement hardware optimizations for 3DES and consequently prefer it over other ciphers.[6]

OpenVPN. OpenVPN is a popular open-source VPN solution originally written by James Yonan. The default encryption for the transport protocol of OpenVPN is Blowfish, a 64-bit cipher, in CBC mode. OpenVPN supports two different ways of generating session keys to encrypt the messages. In pre-shared-key mode, static keys are used for all the traf-

[2] https://wiki.mozilla.org/Security/Server_Side_TLS
[3] https://blog.binaryedge.io/2015/11/10/ssh/
[4] https://support.microsoft.com/en-us/kb/325158
[5] http://www.jacco2.dds.nl/networking/openswan-macosx.html
[6] http://www.cisco.com/c/en/us/td/docs/ios/12_2/12_2z/12_2zj/feature/guide/gtaimvpn.html

Firefox Version				AES	
#	Release date	RC4	3DES	CBC	GCM
35	2015/01/13	20.4	0.15	27.5	51.9
36	2015/02/24	0.20	1.81	45.5	52.5
37	2015/03/31	0.13	1.81	41.7	56.4
38	2015/05/12	0.09	1.82	36.0	62.1
39	2015/07/02	0.08	1.81	31.6	66.5
40	2015/08/11	0.08	1.68	29.7	68.6
41	2015/09/22	0.08	1.44	27.9	70.6
42	2015/11/03	0.08	1.26	27.9	70.7
43	2015/12/15	0.05	1.19	28.6	70.2
44	2016/01/26	–	1.17	28.3	70.5
45	2016/03/08	–	1.13	28.1	70.8

Table 2: Cipher use for TLS connection from the Firefox web browser (percentage). The figures come from telemetry data available at https://telemetry.mozilla.org/ (we use the `SSL_SYMMETRIC_CIPHER_FULL` metric), and are gathered during the lifetime of a particular version. RC4 is fallback-only since version 36.

fic. In particular, there is no limit to the lifetime of those keys. In TLS mode, session keys are generated with a TLS handshake, using certificates to authenticate the peers. The session keys are updated periodically, with limits on the number of packets, the number of bytes, or a session time. The default configuration rekeys the tunnel every hour. In addition, the tunnel is rekeyed shortly before 2^{32} packets in TLS mode, because the packet counter is a 32-bit integer. In pre-shared-key mode, the packet counter is a 64-bit integer.

3.3 3DES usage in HTTPS

3DES is the second most widely supported cipher (after AES) in HTTPS servers, with about 87% of servers supporting it.[7] Furthermore, all popular web browsers support 3DES. The cipher that is actually negotiated for a TLS connection is chosen by the server, based on its local preference order and on the order in which the client advertises its ciphersuites. Since most modern browsers and servers prefer AES over 3DES, we find evidence that 1-2% of all TLS connections likely use 3DES in CBC mode, as detailed below.

Browser Telemetry. Telemetry data from Mozilla Firefox (given in Table 2) show that 3DES is used for more than 1% of HTTPS connections from Firefox browsers. Interestingly, the use of 3DES with Firefox has actually increased recently, as a consequence of the removal of RC4 from the list of supported ciphers in version 36 (following [3] and RFC 7465). Indeed a number of servers are configured to use, in order of preference, first RC4, then 3DES, and now use 3DES with Firefox.

Other browser vendors have privately reported the following telemetry data to us. Microsoft sees 1.2% 3DES connections for Internet Explorer and Edge, but only 0.8% system wide (the lower value can be explained by the fact that some applications still support RC4). Google Chrome sees 1.1% of connections using 3DES, but this number peaked at 1.6% soon after RC4 was disabled.

[7] https://securitypitfalls.wordpress.com/2016/04/17/february-2016-scan-results-incomplete/

3DES support	Andr5	FF44	IE11	Chr47
Top 1k	93%	0.4%	1.6%	1.6%	1.6%
Top 10k	92%	1.0%	2.1%	2.1%	2.1%
Top 100k	89%	0.8%	1.9%	1.9%	1.9%
Top 1m	86%	0.6%	1.3%	1.3%	1.3%

Table 3: 3DES support for HTTPS servers in Alexa's top website list. We count servers that support 3DES, and servers that would select 3DES with common browsers: Android 5.0 integrated browser, Firefox 44, Internet Explorer 11, and Chrome 47. Scans performed on February 16 – 18, 2016.

Scanning the Top 1M Websites. We performed a scan of the top 1 million servers as listed by Alexa using the cipherscan tool[8]. We found that 86% of the servers that support TLS include 3DES as one of the supported ciphers. Moreover, using the ciphersuites sent by a modern client (Firefox 44, IE 11 or Chrome 47), we estimate that 1.2% will actually use a 3DES based ciphersuite. We report detailed figures in Table 3.

Windows XP Clients and Windows 2003 Servers. The Windows XP and Windows Server 2003 operating systems do not support AES-based ciphersuites, although support for AES can be added with an optional hotfix. With the default configuration, they support only RC4, 3DES, DES, and RC2-40. While these operating systems are not supported anymore by Microsoft, they still have users, and this creates situations where the best available cipher is 3DES.

If a client uses Internet Explorer on Windows XP or Windows Server 2003 to connect to a modern web server (with no RC4 ciphersuites), the connection will use 3DES or fail. Indeed, supporting Windows XP clients is the main stated reason for servers to continue supporting RC4 and 3DES.

If a web server runs on IIS 6.0 (*i.e.* Windows XP or Windows Server 2003), it will offer only RC4 and 3DES ciphersuites, unless support for AES ciphersuites has been added. If a modern web browser visits the website over HTTPS, the connection will use 3DES. According to web server surveys[9], IIS 6.0 powers about 1.9% of SSL enabled web servers, but it is likely used even more within corporate networks, and these servers would not appear in public scans.

Long-lived HTTPS connections. In order to mount the attack in practice, we need to find client and servers that not only negotiate the use of 3DES, but also exchange a large number of HTTP request in the same TLS connection (without rekeying). This is possible using a persistent HTTP connection, as defined in HTTP/1.1 (Keep-Alive).

On the client side, all browsers that we tested (Firefox, Chrome, Opera) will reuse a TLS connection as long as the server keeps it open. On the server side, we found that a number of HTTP servers will close the TLS connection even when it is still active. In particular, Apache and Nginx limit the number of requests sent in the same connection, with a maximum of 100 in the default configuration[10]. On

[8] https://github.com/jvehent/cipherscan

[9] http://www.securityspace.com/s_survey/sdata/201604/servers.html

[10] The setting is called `MaxKeepAliveRequests` for Apache, and `keepalive_requests` for Nginx.

Website	connection closed after
google.com	1 hour†
facebook.com	no limit detected†
youtube.com	no limit detected†
yahoo.com	no limit detected†
live.com	up to one day†
baidu.com	100 requests
amazon.com	100 requests
wikipedia.org	100 requests
qq.com	2000 requests
twitter.com	50 requests

Table 4: Maximum connection lifetime for Alexa's top 10 servers (detected experimentally). The "†" symbol marks servers which accept at least one million requests in a single TLS connection.

the other hand, IIS does not seem to have such a limit. In practice, many high profile servers accept a very large number of requests in a single TLS connection. We made some experiments using a simple Perl script to send simple HTTP requests over a TLS connection, using pipelining to maximize the throughput. As seen in Table 4, we found that about half of the servers in Alexa's top 10 support a large number of requests without rekeying.

For a better estimate of the number of vulnerable servers, we tested servers from Alexa's top 10k that negotiate 3DES with a modern client. We identified 11483 different HTTPS servers[11], and found that 226 of them (1.9%) negotiate 3DES with a modern client. Moreover, 72 of these (0.6% of the total) also accept to keep a connection open for at least 800k requests. Consequently, the duration of the attack is not unrealistic, at least from the viewpoint of browsers and servers, and we estimate that at least 0.6% of HTTPS connections are vulnerable to our attacks.

Downgrading to 3DES. From our measurements, only 1-2% of HTTPS connections use 3DES, and perhaps even fewer of these connections are to servers that allow long-lived TLS connections. However, ignoring the danger posed by 3DES would be dangerous for several reasons. 3DES is typically the second or third most preferred encryption algorithms in HTTPS, so any change in the client or server configurations, say in response to some future attack on AES, would cause a large number of connections to fallback to 3DES. Furthermore, as recent downgrade attacks on TLS (e.g. FREAK) have demonstrated, protocol and implementation flaws can sometimes cause a weaker cipher to be negotiated. For example, if a downgrade attack from TLS 1.2 to SSL 3 becomes possible, then the strongest SSL 3 cipher that can be negotiated may well be 3DES. As an example, we describe a downgrade attack to 3DES that is currently possible on TLS connections. Although it does not enable our full attack, it nevertheless serves as a useful warning for practitioners.

Downgrading False Start. TLS False Start [23] is an optimization to the TLS handshake that reduces its latency. A client implementing False Start will start sending data before receiving the Finished message from the server. This allows an attacker to force the selection of a ciphersuite by tampering with the ServerHello message (he can replace AES by 3DES, for example) and the client will start sending data before verifying that the ciphersuite was really selected by the authentic server. If the client is willing to send millions of requests and gigabytes of data before finishing the handshake, this could allow our attack even against a server that would not select a 3DES ciphersuite. While 3DES ciphersuite are not recommended for use with False Start in the RFC draft [23], it turns out that they are white-listed for use with False Start with Internet Explorer. We have informed the Microsoft security team and they decided to remove 3DES from this white-list.

[11]We tested common sub-domains: `www`, `m`, `secure`, `signin`, `auth`, `account`, `login`

4. ATTACKING AUTHENTICATED HTTP OVER TLS AND OPENVPN

We now demonstrate concrete attacks against authenticated HTTP sessions even when they are secured by TLS or OpenVPN. First, we identify a few examples of secret authentication credentials that are repeatedly sent by the browser on every request. We then show how we can recover these secrets using block cipher collisions.

4.1 HTTP bearer tokens

Cookie-based Sessions. Modern HTTPS websites use a variety of methods to manage authenticated sessions with their clients. The most popular mechanism is secure *cookies* as specified in RFC6265. Once a user has logged in, the server sets a cookie containing a secret value on the user's browser. The browser will then send the cookie on all subsequent requests to the website, implicitly authenticating the user. For example, if domain.com sets a cookie `C=XXXX`, a subsequent HTTPS request from the browser looks like:

```
GET /path/to/file HTTP/1.1
User-Agent: Mozilla/4.0 ...
Host: domain.com
Cookie: C=XXXXX
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Cookies are sensitive, because an attacker who obtains a session cookie can then log in as the user from a different browser. The cookie acts as a *bearer token* that carries the user's delegated credential. Except for its short-lived nature, it has the same security requirements as a password. Indeed, many websites, such as Facebook, use cookies with very long expiry times, making cookie theft even more attractive.

A cookie for a website is normally included in all requests to that website whether that request was initiated by the user, or a script on the website, or even by a different website. Browsers sometimes impose stricter rules for this last category of requests, called cross-origin or cross-domain requests. For example, XmlHttpRequests sent from one domain to another may not have cookies attached. However, cross-domain requests for images or iframes will still send cookies.

HTTP BasicAuth. On corporate networks and even on the web, many websites are protected using the HTTP Basic Authentication mechanism specified in RFC7617. When the server asks for Basic authentication, the browser prompts the user to enter a username and password into a special dialog and then sends this information (in plaintext) as an HTTP header of the form:

`Authorization: Basic dGVzdDoxMjPCow=`

Once a user has entered his login information once, the browser will typically cache this information and use it on all subsequent requests to the server. Notably, even if a different website creates a cross-domain request to the authenticated server, the BasicAuth credentials will be automatically sent by the browser.

BasicAuth credentials contain the user's password and are hence security-critical. One should only use this authentication mechanism over HTTPS but a number of corporate websites use BasicAuth over HTTP, under the assumption that their users can only access the website over a VPN or some other secure connection.

OAuth and OpenID Connect Access Tokens. With the growth in popularity of single sign-on protocols, many websites now allow users to login with third-party credentials, such as OAuth or OpenID Connect tokens, issued by major websites like Google and Facebook. These tokens are effectively bearer tokens that must then be attached to all authentication and authorization requests and are sent either within the URI or in an Authorization header. While the use of these tokens is specific to each website, an attacker can usually repeatedly trigger the sending of these tokens by re-initiating the login sequence. An attacker who steals one of these tokens can then impersonate the user as well as retrieve sensitive data about the user from the login provider.

4.2 The Beastly Attack Scenario

Our attack scenario, depicted in Figure 1, is similar to the setup used in recent attacks on RC4 [3, 19, 32]. The attacker wants to steal some bearer token that is being repeatedly sent by a browser to a website secured with HTTPS, or an HTTP website accessed through a VPN. We assume that the attacker can control some JavaScript on a web page loaded by the user's browser, either by actively tampering with an HTTP response on the wire, or by hosting a malicious website that the user is fooled into visiting. We also assume that the attacker can observe all the encrypted traffic between the target browser and the secure website.

We further assume that the data is encrypted with a 64-bit block cipher in CBC mode (either an HTTPS connection where the client and server have negotiated 3DES, or an HTTP connection though a VPN encrypted with Blowfish or 3DES). Suppose the victim is already logged in to a website and has a session cookie. The attacker runs malicious JavaScript code on the victim's browser that repeatedly sends HTTP queries to the target website server, each containing the session cookie. If he sends close to 2^{32} queries, a collision is expected between a ciphertext block corresponding to the cookie (c_i), and a known block (c_j), containing a known part of the query. The collision attack against CBC reveals the session cookie: $p_i = p_j \oplus c_{i-1} \oplus c_{j-1}$.

Note that most of the traffic generated by the attacker is known or predictable. The requested URL is chosen by the attacker, and all the headers excepted the cookie are predictable and can be observed in any plaintext HTTP traffic. We will assume that the Cookie takes 16-bytes, *i.e.* two blocks, and that the attacker has aligned the cookie to a block boundary using other headers. As explained in Section 2.2, if a query contains 2^u known blocks, the number of encrypted blocks required for the attack is about:

$$\sqrt{H_2} \cdot \sqrt{\pi/2} \cdot 2^{(n+u-1)/2}.$$

In practice, an HTTP query is about 512 bytes (64 blocks), which implies a total complexity of $2^{35.1}$ blocks ($2^{29.1}$ queries), and a total captured trace of size 280 GB.

However, the attack can be made faster by using larger queries: this increases the total amount of data to capture, but it reduces the total number of queries. Since the speed of clients and servers is dependent on the number of queries, a larger query size can result in a faster attack overall. In our experiments, we got good results with 4 kB queries (512 blocks). In this setting, the attack requires $2^{36.6}$ blocks (785 GB), but this corresponds to only $2^{27.6}$ queries: increasing the requests size by a factor 8 (from 512 bytes to 4 kB) increases the total data to capture by a factor $\sqrt{8}$, and reduces the number of queries needed by a factor $\sqrt{8}$. In practice large queries can be produced by adding a long query string to the URL.

4.3 Proof-of-concept Code

The attacker code consists of two parts: a JavaScript program that sends a large number of HTTP requests, and a network adversary who processes the resulting ciphertext to recovers a 16-byte secret.

Man-in-the-browser code. We experimented with hidden `<img>` tags in a web-page with Javascript code to the `onload` event to reload the images repeatedly, and web workers issuing `XmlHttpRequests`. Cookies are automatically inserted when loading images from a different domain, and are also included in cross-domain `XmlHttpRequests` if the `withCredentials` property is set to `true` (this is only allowed for synchronous request, and the request result can not be read, but these restrictions don't prevent our attack). According to our experiments, request are generated faster using `XmlHttpRequest`. In order to avoid caching, we request a non-cacheable resource. Alternatively, we can include a unique query string in each URL, but the responses are still cached by the client and this can affect the query rate. We experimented with several browsers, and we obtained the best results with Firefox Developer Edition 47.0a2 [12].

Recovering collisions. We captured the encrypted packets with `tcpdump` and used a C++ program to extract the ciphertext blocks (using `libpcap`). In both the HTTPS attack and the OpenVPN attack, each HTTP query is sent in a separate encrypted record, which contain the plaintext at a fixed position, as well as some extra information (packet number, padding, MAC, ...). Therefore, it is easy to know to which plaintext block corresponds each ciphertext block, and to align the cookie to a block limit.

After capturing all the traffic, the C++ program sorts the ciphertext blocks in order to locate collisions. Since the amount of data is quite large (hundreds of gigabytes), we use the external sort implementation of the `stxxl` library. With a NAS storage, sorting the data took around four hours.

4.4 Attacking Basic Auth over OpenVPN

To demonstrate the attack against OpenVPN, we use a pre-shared-key tunnel between two physical machines running Linux, with Firefox Developer Edition 47.0a2 on one side, and an nginx server on the other side. Access to the server is protected by BasicAuth, and the user has entered his

[12] We have similar results with the stable version Firefox 48.0 which was released after we made our experiments.

credentials. Using the default OpenVPN settings, the tunnel is encrypted with Blowfish in CBC mode.

We use the Javascript code described in the previous section to send a large number of requests to the server though the tunnel. We found that increasing the size of the request to 4 kB does not significantly reduce the query rate, but reduces the number of queries required for the attack. In our setting, the browser generates about 2900 requests per second, using several web Worker running in parallel. We expect the first collision after roughly $2^{32.3}$ blocks (40 GB), *i.e.* one hour. In practice, we detected the first collision early, after only 30 minutes ($2^{31.3}$ blocks); as expected, the predicted xor difference was correct. The full attack needs about $2^{36.6}$ blocks (785 GB) to recover a two-block secret out of 4 kB messages; this should take about 19 hours in this setting. In our demo, it took 18.6 hours and 705 GB, and we successfully recovered the 16-byte authentication token.

OpenVPN in TLS mode. In TLS mode, OpenVPN rekeys the tunnel every hour with the default settings (as apposed to the pre-shared-key mode used above that doesn't do any rekeying). With our experimental setting, this means we can only capture about $2^{23.3}$ requests encrypted with a given key, *i.e.* $2^{32.3}$ blocks. Therefore, we expect to detect a collision with probability close[13] to 1/2. In order to recover a two-byte secret out of 4 kB messages, we need about 384 collisions as explained in Section 2.2. Therefore, this attack will take roughly 768 hours, or 32 days.

This illustrate the point that keys should be changed *well before* reaching 2^{32} blocks: if each session is broken after 2^{32} blocks, there is still a collision with probability roughly $1-e^{-1/2} \approx 0.4$. This increases the time required by the attack to 40 days, but it will still succeed with high probability.

4.5 Attacking Secure Cookies over HTTPS

The attack against HTTPS connection is very similar to the attack against OpenVPN, as long as we have a client and server that negotiate 3DES, and that keep a connection open for a long time.

Vulnerable Websites. As detailed in Section 3.3, about 0.6% of the top Alexa 100K websites support 3DES and allow long-lived HTTPS connections. This list contains many high-profile e-commerce and banking websites, including the ones shown in Table 5. In addition, we found a number of domains were the server serving `domain.com` will use 3DES, but it just redirects the client to a better configured server for `www.domain.com`. This could still open the way for our attacks, if sessioncookies for `www.domain.com` are set at the top-level domain and hence also sent to `domain.tld`. The last three websites in Table 5 belong to this category.

Proof-of-Concept Attack Demo. In our setup we use Firefox Developer Edition 47.0a2 running on Linux and an IIS 6.0 server in a Windows Server 2003 R2 SP2 Virtual machine. The default configuration of IIS 6.0 with all recommended updates offers only RC4 and 3DES ciphers, and will use 3DES with Firefox and other recent browsers where RC4 is disabled. Moreover, IIS 6.0 supports HTTP/1.1 and keeps an active connection open for an arbitrary long time.

We use the Javascript code described in Section 4.3, but we note that with several `Workers` running in parallel (or several `<img>` tags), modern browsers open a few parallel connections to the server (typically, 6) and split the requests over these connections. For our attack, we need to maximize the throughput over a *single* connection. This can be achieved by disturbing some of the connections so that most of request are send in a single. In our setup, we used `iptables` rules to limit the rate of all connections except one. In a real attack, this would be done by an active man-in-the-middle, but a passive man-in-the-middle can also mount the attack – it will just take more time to collect the data.

[13]The probability can be computed as $1 - e^{-2^{32.2}2^{32.2}/2^{64}}$.

Website	Category
https://signin.ebay.com	E-commerce
https://account.nasdaq.com	Finance
https://www.bancomercantil.com	Banking
https://www.unionbankonline.co.in	Banking
https://ziraatbank.com.tr	Banking
https://www.state.nj.us	Government
https://secure.match.com	Dating
https://amadeus.net	Travel
https://walmart.com	Corporate
https://citrix.com	Corporate

Table 5: High-value websites that negotiate 3DES with modern browsers, and accept at least 1 million requests in the same TLS session

On Firefox Developer Edition 47.0a2, with a few dozen `Workers` running in parallel, we can send up to 2000 requests per second in a single TLS connection. To further reduce the time needed for the attack, we inject a padding cookie to expand the requests to 4 KB (512 blocks). In our setting this reduces the rate to 1500 requests per second, but it still leads to a faster attack. Again, we expect the first collision after roughly $2^{32.3}$ blocks (40 GB), *i.e.* $2^{23.3}$ queries. This should take slightly less than two hours.

In our experiment, we were lucky to detect the first collision after only 25 minutes ($2^{20.1}$ requests), and we verified that the collision revealed the xor of two plaintexts blocks. As seen previously, the full attack should require $2^{36.6}$ blocks (785 GB) to recover a two-block cookie, which should take 38 hours in our setting. Experimentally, we recovered the two-block cookie after only 30.5 hours and 610 GB.

Note that special care must be taken to recover the first block of a TLS fragment. When a collision involves the first ciphertext block ($c_i^{(j)} = c_0^{(j')}$), this gives an equation involving the IV ($c_{i-1}^{(j)} = c_{-1}^{(j')}$). In particular, this requires special treatment for TLS 1.0, because the IV of a given of a given record is not explicitly included in the message: it is the last ciphertext block of the previous record.

Our attack above relied on knowing most of the plaintext, but if we have enough data we can even recover most of the plaintext using just a single known block. In our experiment, after 65 hours ($2^{28.3}$ requests, 1.3TB) we can recover 492 blocks out of 510, assuming a single known plaintext block.

5. IMPACT AND MITIGATION

We have demonstrated the first concrete attacks on mainstream Internet protocols that exploit block cipher collisions. Our attacks can recover valuable secrets such as HTTP cookies and passwords in under 40 hours. Our attacks impact a

majority of OpenVPN connections and an estimated 0.6% of HTTPS connections to popular websites. We expect that our attacks also impact a number of SSH and IPsec connections, but we do not have concrete measurements for these protocols. Like many recent attacks on TLS, such a BEAST and RC4 NOMORE, the underlying principles behind our attacks were well known to cryptographers. Our goal is to raise awareness among practitioners about the vulnerabilities of short block ciphers and on safe ways of using them.

5.1 Comparison with RC4 attacks

Our attack scenario is very similar to the setup of the recent attacks on the use of RC4 in HTTPS. We use the same man-in-the-browser setting to generate a large number of HTTP requests, and the data complexity of our attacks is comparable to these prior works:

- the first attack by Al Fardan *et al.* [3] uses between 2^{28} and 2^{32} sessions, with rekeying for each session, to recover the first 220 bytes a the message (but the cookie is not usually in these 220 bytes);
- the second attack by Al Fardan *et al.* [3] uses between 2^{33} and 2^{34} request, in the same TLS session or in different sessions, to recover a cookie;
- Garman *et al.* [19] improved the first attack to require only 2^{26} sessions when targeting a BasicAuth password, using a better guessing strategy;
- the latest attack by Vanhoef and Piessens [32] requires $2^{30.2}$ requests, in the same TLS session or in different sessions, to recover a cookie.

In comparison, our attack requires only $2^{29.1}$ short queries of 512 bytes (280 GB in total), which can be reduced to $2^{27.6}$ longer queries of 4 kB (785 GB in total). However, these numbers are for the case when all the data is encrypted within the same session. Even if the amount of data sent on a single connection is limited, as long as the limit is close enough to the birthday bound, we can still mount our attacks across multiple parallel and sequential sessions, albeit with a higher data and time complexity.

5.2 Mitigation

The obvious way to avoid these attacks is to stop using legacy 64-bit block-ciphers. There is no good reason to configure a VPN or an HTTPS server to prefer 3DES (or Blowfish) over AES. AES is more secure, is a FIPS standard, and will almost certainly be faster: it is implemented in hardware in most server and desktop CPUs, and can be implemented efficiently in dedicated hardware when an extremely high throughput is required.

On the other hand, there are specific uses of cryptography were 128-bit block ciphers can not be used, because of legacy reasons, or due to the relatively large hardware footprint of 128-bit ciphers (in this case, *lightweight* 64-bit block ciphers such as PRESENT or HIGHT would typically be used). In these scenarios, protocol designers should pay close attention to the issue of birthday attacks. In particular, they should verify that the amount of data encrypted with a fixed key is *significantly* smaller than 2^{32} blocks, or use modes that provide security beyond the birthday bound, such as CENC [22] or PMAC_plus [33].

Concretely, we recommend the following measures to prevent our attack:

1. Web servers and VPNs should be configured to prefer 128-bit ciphers. According to our scans, about 1.1% of the top 100k web server from Alexa, and 0.5% of the top 1 million, support AES but prefer to use 3DES.

2. Web browsers should offer 3DES as a fallback-only cipher, to avoid using it with servers that support AES but prefer 3DES.

3. TLS libraries and web browsers and servers should limit the length of TLS sessions with a 64-bit cipher. This could be done in TLS renegotiation, or in some cases by closing the connection and starting a new one (*i.e.* limiting HTTP/1.1 Keep-Alive, SPDY, and HTTP/2 with 3DES ciphersuites).

5.3 Responsible Disclosure

We have communicated our results and concerns to the OpenVPN team, and to various website owners, browser vendors, and TLS libraries. They all acknowledged the issue, and are working on implementing countermeasures. The TLS vulnerability received CVE number CVE-2016-2183.
OpenVPN will display a warning to users who choose to use 64-bit ciphers and encourage them to transition to AES-GCM (cipher negotiation is also being implemented in the 2.4 branch). It will also implement a default renegotiation limit of 64MB when used in TLS mode.
OpenSSL has moved 3DES ciphersuites from the `HIGH` category to `MEDIUM` in the 1.0.2 branch, and will disable it by default in the upcoming 1.1.0 release.
Akamai will offer an option for web server administrators to drop 3DES from the offered ciphers.
Apple has disabled 3DES on icloud.com and is recommending that all its customers disable 3DES on their websites.

Currently, most browsers see about 1% of their connections using 3DES, and vendors consider this number too high to simply disable 3DES on the client side, since too many websites would be broken. So, they are instead considering implementing data limits per connection to force rekeying, or offering 3DES ciphersuites only in a fallback negotiation if no AES ciphersuite is acceptable to the server.
Mozilla is implementing data limits for all ciphersuites: https://bugzilla.mozilla.org/show_bug.cgi?id=1268745 .
Microsoft has removed 3DES from the False Start whitelist: https://technet.microsoft.com/library/security/3155527.aspx
More details about implemented countermeasures will be added to our webpage as they become available:
http://sweet32.info

6. REFERENCES

[1] M. Abdalla and M. Bellare. Increasing the lifetime of a key: a comparative analysis of the security of re-keying techniques. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 546–559. Springer, Heidelberg, Dec. 2000.

[2] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Z. Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 5–17. ACM Press, Oct. 2015.

[3] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt. On the Security of RC4 in TLS. In S. T. King, editor, *USENIX Security*, pages 305–320. USENIX Association, 2013.

[4] N. J. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540. IEEE Computer Society Press, May 2013.

[5] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Kasper, S. Cohney, S. Engels, C. Paar, , and Y. Shavitt. DROWN: Breaking TLS using SSLv2, 2016. https://drownattack.com.

[6] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, Oct. 1997.

[7] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In Y. Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 341–358. Springer, Heidelberg, Aug. 1994.

[8] M. Bellare, T. Kohno, and C. Namprempre. The Secure Shell (SSH) Transport Layer Encryption Modes. IETF RFC 4344, 2006.

[9] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552. IEEE Computer Society Press, May 2015.

[10] K. Bhargavan and G. Leurent. Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH. In *ISOC Network and Distributed System Security Symposium (NDSS16)*, 2016.

[11] K. K. Bodo Moller, Thai Duong. This POODLE Bites: Exploiting The SSL 3.0 Fallback, 2014. https://www.openssl.org/~bodo/ssl-poodle.pdf.

[12] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In P. Paillier and I. Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, Heidelberg, Sept. 2007.

[13] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. IETF RFC 5246, 2008.

[14] W. Diffie and M. E. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, 1979.

[15] T. Duong and J. Rizzo. Here come the $\oplus$ ninjas. *Unpublished manuscript*, 2011.

[16] M. Dworkin. Recommendation for Block Cipher Modes of Operation. NIST Special Publication 800-38{A,B,C,D}, National Institute for Standards and Technology, 2001 – 2011.

[17] P. Erdos and A. Rényi. On the evolution of random graphs. *Bull. Inst. Internat. Statist*, 38(4):343–347, 1961.

[18] S. Frankel and S. Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. IETF RFC 6071, 2011.

[19] C. Garman, K. G. Paterson, and T. V. der Merwe. Attacks Only Get Better: Password Recovery Attacks Against RC4 in TLS. In J. Jung and T. Holz, editors, *USENIX Security*, pages 113–128. USENIX Association, 2015.

[20] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee. HIGHT: A new block cipher suitable for low-resource device. In L. Goubin and M. Matsui, editors, *CHES 2006*, volume 4249 of *LNCS*, pages 46–59. Springer, Heidelberg, Oct. 2006.

[21] On the Assessment of Cryptographic Techniques and Key Lengths, 4th edition. ISO/IEC JTC 1/SC 27 Standing Document 12, May 2014. Available online: http://www.din.de/blob/78392/6f4bbd95d0cf11d1b32784948039600b/sc27-sd12-data.pdf.

[22] T. Iwata. New blockcipher modes of operation with beyond the birthday bound security. In M. J. B. Robshaw, editor, *FSE 2006*, volume 4047 of *LNCS*, pages 310–327. Springer, Heidelberg, Mar. 2006.

[23] A. Langley, N. Modadugu, and B. Moeller. Transport Layer Security (TLS) False Start. Internet Draft, Nov. 2015. https://tools.ietf.org/html/draft-ietf-tls-falsestart-01.

[24] A. Luykx and K. G. Paterson. Limits on authenticated encryption use in TLS, march 2016. http://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf.

[25] D. McGrew. Impossible plaintext cryptanalysis and probable-plaintext collision attacks of 64-bit block cipher modes. In *Fast Software Encryption Workshop (FSE)*, 2013. https://eprint.iacr.org/2012/623.

[26] D. McGrew and P. Hoffman. Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH) . IETF RFC 7321, 2014.

[27] E. Petrank and C. Rackoff. CBC MAC for real-time data sources. *Journal of Cryptology*, 13(3):315–338, 2000.

[28] J. Rizzo and T. Duong. The crime attack. In *EKOparty Security Conference*, volume 2012, 2012.

[29] P. Rogaway. Problems with Proposed IP Cryptography. Unpublished draft, 1995. http://web.cs.ucdavis.edu/~rogaway/papers/draft-rogaway-ipsec-comments-00.txt.

[30] P. Rogaway. Evaluation of Some Blockcipher Modes of Operation. Technical report, CRYPREC, Feb 2011.

[31] P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[32] M. Vanhoef and F. Piessens. All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS. In J. Jung and T. Holz, editors, *USENIX Security*, pages 97–112. USENIX Association, 2015.

[33] K. Yasuda. A new variant of PMAC: Beyond the birthday bound. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 596–609. Springer, Heidelberg, Aug. 2011.

[34] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. IETF RFC 4253, 2006.