

Moving to client-side hashing for online authentication

Enka Blanchard, Xavier Coquand, Ted Selker

▶ **To cite this version:**

Enka Blanchard, Xavier Coquand, Ted Selker. Moving to client-side hashing for online authentication. 9th International Workshop on Socio-Technical Aspects in SecuriTy, Sep 2019, Luxembourg Ville, Luxembourg. hal-02560695

HAL Id: hal-02560695
https://univ-paris8.hal.science/hal-02560695

Submitted on 2 May 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Moving to client-side hashing for online authentication

Enka Blanchard[1], Xavier Coquand[2], and Ted Selker[3]

[1] Digitrust, Loria, Université de Lorraine
Nikola.K.Blanchard@gmail.com, www.koliaza.com
[2] Bsecure, Paris
[3] University of Maryland, Baltimore County

Abstract. Credential leaks still happen with regular frequency, and show evidence that, despite decades of warnings, password hashing is still not correctly implemented in practice. The common practice today, inherited from previous but obsolete constraints, is to transmit the password in cleartext to the server, where it is hashed and stored. We investigate the advantages and drawbacks of the alternative of hashing client-side, and show that it is present today exclusively on Chinese websites. We also look at ways to implement it on a large scale in the near future.

Keywords: Hashing · Web standards · Authentication

1 Introduction

Despite multiple decades of insistence from the cybersecurity and cryptography communities, password hashing is still far from a solved problem in practice. Two issues are even more critical today than they were more than 25 years ago, when vulnerabilities were first found in MD5. The first is that, although it was first mentioned as important to security in the 1960s, long before the existence of the Internet, password hashing is still not as commonplace as it should. Many recent database leaks with passwords in clear reveal that even some of the largest service providers still do not follow what was already best practices when they were created. The most recent example is Facebook's revelation in March 2019 that they kept a log file with more than 200 million cleartext passwords that was accessible by more than 2000 developers. The second issue is that hashing techniques have changed, and distributed computation on specialised hardware has made many hashing algorithms obsolete for password purposes. Well applied modern hashing techniques are still exceedingly rare, with the only major leaks that showed this level of security coming from online password managers such as LastPass [42,22].

There are many explanations for such problems, most of them with a social component. First, developers who implement the security procedures do not always have the relevant training [14,2]. This is linked to a culture of going faster,

at the expense of good security practices [1]. This, in turn, comes from the fact that service providers generally suffer from negative outcomes only when security breaches become public. Even in such cases, the incentives are not always strong enough to effect real change — Yahoo! suffered from three leaks of increasing magnitude between October 2016 and October 2017, which could have been prevented had security been reinforced after the first [41].

There are many ways to address this issue, but blaming the developers has not worked so far [14]. As such, we investigate the possibility of client-side password hashing to be an alternative to the standard practice of server-side hashing. Its main advantage is that client-side hashing, as opposed to server-side, is easily detectable and analysable. This creates accountability, means that it becomes possible to impose a direct cost on companies with poor security practices. Thus, we can give a strong incentive to companies to reform their practices before they suffer from major public security breaches.

This paper is organised as follows. We start by looking at the current state of the art, both on how to hash securely and how current service providers fail to do so. We then turn to client-side hashing, by looking at how prevalent it is before listing its advantages and drawbacks. We finish by looking at how large scale changes could be made in the near future, and discussing potential improvements and complications.

2 Password hashing today

Problems were first found with the practice of storing all the passwords unencrypted in a single text file in the 1970s [37]. The general password architecture that was developed at the time has not evolved much in the decades since. The best practice still consists in hashing the password with a salt, storing this hash on the server, and comparing the hashes to make a decision when the user tries to login again. Client-based hashing was not always a possibility, due to compatibility issues with legacy protocols [35]. This is not true anymore — although there is still a generally unspoken assumption that all hashing should be done server-side. The server hashing cost was even a major point of contention in a recent password hashing algorithm competition [20], without the authors mentioning the possibility and impact of client-side hashing.

2.1 Best practices

As the general architecture has not changed much in the past decades, the main questions are still the same: Which hashing function to use? How to salt the password? How to prevent side-channel attacks?

Hashing algorithm. After problems were found in the first two common hashing algorithms — MD5 and SHA-1 — the two principal general-purpose hashing algorithms currently considered to be secure are SHA-2 and SHA-3 — or rather, algorithms from these families. However, this has a caveat. They are secure

insofar as the string to hash has high entropy, in which case finding a preimage for the function becomes a hard problem. Hashing a complete message with these algorithms is then secure. However, if the number of possibilities for the original message is limited, one can use brute-force and hash all the potential messages. Unlike online attacks where 10^{10} bogus login attempts are easily noticeable, it is easy to compute that many hashes offline, on special-made machines. This is one of the main risks with database leaks, as even consumer machines can nowadays brute-force more than 1 billion such hashes per second [15,11,43].

With that kind of capability, dictionary-augmented brute-forcing becomes a strong option, as most passwords deviate only poorly from real words. This is true no matter the kind of hashing algorithm used: if the space of all possible inputs remain small and the search can be parallelised efficiently, it is enough to hash all probable inputs to get the inverse of most hashes. To address this, two main algorithms were initially used, PBKDF2 and bcrypt, which can run recursively n times on their own output to artificially increase the computing time [49]. This slows down the algorithm in an unavoidable way, negating the gains due to more powerful machines but is still vulnerable to parallel brute-forcing. Argon2 is a more recent solution, specifically made to be hard to parallelise by requiring an arbitrarily large amount of memory [8], and is now one of a set of good alternatives [20]. All the alternatives provide an adjustable parameter to increase the time and space complexity of hashing.

Salt. The second common component in the hashing process is the *salt* — a pseudorandom string that is concatenated with the password before hashing. This is done to prevent attacks that make use of a large table of precomputed hashes, also known as a rainbow table [33]. The salts should be different from user to user — which is not always understood by service providers. If a single salt is used for a whole database, it prevents attacks using generic rainbow tables, but allows the computation of a website-specific rainbow table, only marginally improving security. Salts are often used as an example of data stored only on the server's side. However, in this context of password hashing, the only goal of the salt is to prevent precomputed tables. As such, the salt can just be the login and website name, which is unique and always available client-side. When using one service to login into another, one must however be careful and make sure that the right website name is used.

Side-channels. Even using secure hashing with unique salts is not enough if adversaries have opportunities to obtain the passwords through other means. There is a large variety of side-channel attacks that are relevant, but two are particularly important. The first lies in stealing the password itself when it is transmitted in cleartext, but that has been thankfully made mostly obsolete by the switch from HTTP to HTTPS [47]. The second comes from unsecure password management on the server side, for example the storage of log files with incoming requests (including cleartext passwords), as was done until recently by Facebook [32]. Never storing sensitive data is a first step towards security, with the advantage of being easier than never receiving it in the first place.

PAKE. There already exists an alternative protocol that addresses the issues of server-side hashing: Password Authenticated Key Exchange (PAKE) [7] , and its derivatives, among which the Secure Remote Password protocol (SRP) [50] is the best known. This protocol integrates asymmetric cryptography and ideas from zero knowledge protocols to prevent the server from having enough information to independently recomputing the password without mounting costly brute-force attacks.

Various problems have plagued different PAKE implementations and prevented widespread use, among which we can mostly cite patent problems, as well as security issues in earlier versions of SRP [17], as it is already two decades old. The main issue, however, is that it is quite a complex protocol, and cannot be implemented as easily as a simple hashing function. Some more modern alternatives exist, such as OPAQUE [25], but they are still far from being commonly used.

2.2 Recent database leaks

Credential leaks are becoming increasingly commonplace, with weekly reports of stolen credentials [41], not only from start-ups and smaller corporations but also from the biggest companies. As vulnerabilities in both MD5 and SHA-1 have been public for more than a dozen years, one could expect that most service providers would update their policies (even if hashing server-side), but this is sadly not the case. Because of this, some leaks reach catastrophic proportions, as can attest the discovery in mid-March 2019 that Facebook had stored between 200 and 600 million passwords in cleartext instead of hashing, going as far back as 2012 [32]. Facebook revealed that the stored passwords were only accessible to employees — and were accessed by about 2000 of them — leaving open the question of why they had stored them in the first place. Less than a month later, it was revealed that the social network had also asked some of its users to provide their login details for their main email addresses, breaching all forms of privacy concerns [27]. This is not a freak occurrence, as Twitter and GitHub both revealed similar failures to encrypt their confidential information in the previous ten months [31,48].

In an extensive analysis [24], Jaeger et al. looked at 31 credential leaks going from 2008 to 2016, totalling close to 1 billion email/password pairs worldwide. Of the leaks considered, more than half consisted of entirely unencrypted stored credential pairs (including gmail.com in 2014 and twitter.com in 2016, although they could not make sure the data was authentic), and only one, ashleymadison.com, used a strong level of encryption — bcrypt — while still making some mistakes. The main mistake made was storing MD5 hashes of case-insensitive versions of the passwords, from which it was possible to compute a preimage, leaving the option of computing the full password by hashing the 1000 or so remaining possibilities through bcrypt [16]. Table 1, partially extracted from [24], shows the main authenticated leaks they analysed.

website	encryption	# accounts leaked	leak date
myspace.com	SHA-1	358986419	2008
gawker.com	DES	487292	Dec. 2010
aipai.com	MD5	4529928	Apr. 2011
csdn.net	clear	6425905	Oct. 2011
tianya.cn	clear	29642564	Nov. 2011
vk.com	clear	92144526	2012
linkedin.com	SHA-1	112275414	Feb. 2012
imesh.com	MD5+salt	51308651	Sep. 2013
xsplit.com	SHA-1	2990112	Nov. 2013
51cto.com	MD5+salt	3923449	Dec. 2013
xiaomi.com	MD5+salt	8281358	May 2014
000webhost.com	clear	15035687	Mar. 2015
sprashivai.ru	clear	3472645	May 2015
ashleymadison.com	bcrypt	36140796	July 2015
17.media	MD5	3824575	Sep. 2015
mpgh.net	MD5+salt	3119180	Oct. 2015
r2games.com	MD5+salt	11758232	Oct. 2015
nexusmods.com	MD5+salt	5918540	Dec. 2015
mate1.com	clear	27402581	Feb. 2016
naughtyamerica.com	MD5	989401	Apr. 2016
badoo.com	MD5	122730419	June 2016

Table 1. Partial list of leaks analysed by Jaeger et al. with number of credentials leaked, date and encryption method used in each case, extracted from [24].

One common problem with this list is that we can only discover that service providers were using obsolete security techniques after the damage is done, or even much later if they do not immediately disclose observed breaches [26]. This is where client-side hashing comes into play, as it is much easier to detect.

3 Detecting client-side hashing

One of the main interests of client-side hashing is that it is observable by the user. Detecting it, however, often requires work. Some service providers still rely on security through obscurity, and make their scripts obfuscated to make attacks harder. Except in rare cases, passwords are by now generally encrypted (with a symmetric encryption algorithm) before leaving the client's machine. As such, checking whether the password is still visible in outgoing packets would only catch the very worst cases, where the password is neither hashed nor encrypted. Thankfully, there are at least two different methods to check whether *sufficiently secure* hashing is being performed.

3.1 Syntactic and semantic analyses.

The first method is the most precise of the two, and relies on — potentially automated — code analysis. One of the simplest way is to check the libraries

called by the current webpage and infer from them (for example, the presence of no hashing library besides the inclusion of an MD5 function would be a red flag). An improvement would be to automatically detect the password field and follow the path of the relevant memory object (or to check whether any object sent in an outgoing packet is identical to the password). As it depends on the skill of the person analysing the code, this is the most versatile method and can even work with custom-made hashing methods, but cannot be entirely automated. It also struggles against hashing that relies on compiled code.

3.2 Computing load analysis.

An alternative and more efficient method could be used in the near future to detect whether the website implements client-side hashing, and whether it is secure enough. One issue is that it is not immediately relevant, as the proportion of websites that would currently be considered secure would be infinitesimal. The idea is quite simple: any correct implementation of a secure password hashing algorithm requires a surge in memory and processor usage. Detecting it would be doable, although a surge could be linked to a different process. As such, it can mostly be used to quickly detect websites where the hashing is visibly insufficient. Both methods could also be combined to indicate a failure to correctly hash in most dangerous cases — while proving that it is correctly hashed would still be harder.

3.3 Manually checking the Alexa top 50

We decided to use manual semantic analysis to check which of the top 50 global websites — according to Amazon Alexa [4] — implemented client-side hashing. Table 2 shows the results of this small experiment. Figure 1 shows an example of cleartext password sent to the server (using TLS, but no hashing) on facebook.com and the equivalent on baidu.com.

Analysis of the websites with client-side hashing. Out of the top 50 websites, we only found 8 with client-side hashing. This is slightly misleading, however, as some of the concerned websites, including 360.cn and qq.com, use the same authentication system, made by baidu.com. Other websites — like csdn.net and taobao.com — do not redirect to baidu.com but reuse very similar authentication templates. Moreover, the 8 websites with client-side hashing correspond exactly to the 8 websites from the top 50 that are based in the People's Republic of China. There are different potential explanations, which we will now investigate.

3.4 Why is client-side hashing rare?

This question we ask is twofold. First, why does every Chinese website implement client-side hashing, and second, why are they the only ones to do so? Alas, we do not have access to the decision-making process that led to this state of affairs. However, we can make informed guesses by looking at regulations and incentive structures.

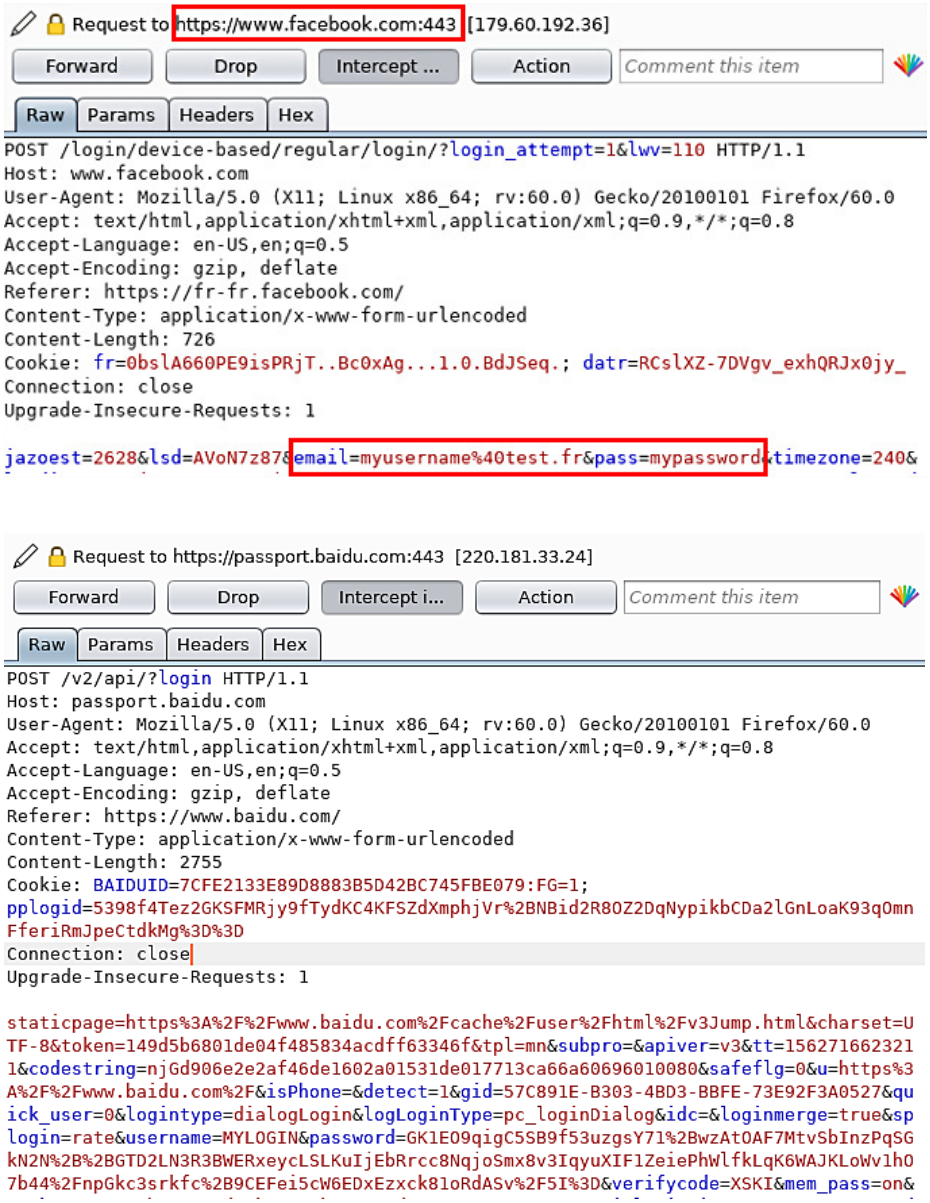


Fig. 1. Request sent to facebook.com (top) and baidu.com (bottom) by TLS after clicking on the login button. For facebook.com, the cleartext password is shown on the bottom line, bordered in red. For baidu.com, the encrypted password is shown on the third line from the bottom, right after the username.

Website	Client-side	Website	Client-side
google.com	NO	youtube.com	NO
facebook.com	NO	**baidu.com**	**YES**
wikipedia.org	NO	**qq.com**	**YES**
yahoo.com	NO	amazon.com	NO
taobao.com	**YES**	twitter.com	NO
tmall.com	NO	reddit.com	NO
instagram.com	NO	live.com	NO
vk.com	NO	sohu.com	NO
jd.com	NO	yandex.ru	NO
sina.com.cn	**YES**	**weibo.com**	**YES**
blogspot.com	NO	netflix.com	NO
linkedin.com	NO	bilibili.com	NO
twitch.tv	NO	pornhub.com	NO
login.tmall.com	NO	**360.cn**	**YES**
csdn.net	**YES**	yahoo.co.jp	NO
mail.ru	NO	bing.com	NO
microsoft.com	NO	whatsapp.com	NO
naver.com	NO	aliexpress.com	NO
livejasmin.com	NO	microsoftonline.com	NO
alipay.com	**YES**	ebay.com	NO
xvideos.com	NO	tribunnews.com	NO
amazon.co.jp	NO	google.co.in	NO
github.com	NO	okezone.com	NO
imdb.com	NO	google.com.hk	NO
pages.tmall.com	NO	stackoverflow.com	NO

Table 2. Result of a manual analysis on which websites implement client-side hashing. A YES was given to each website where the password was not simply symmetrically encrypted using TLS. All websites come from the Alexa Top 50 global websites on 07-07-2019, with the left column corresponding to ranks 1-25, and the right one to ranks 26-50.

Chinese client-side hashing. The PRC imposes strong constraints on the type of cryptography that can be used on its territory and by its companies [34], so it is normal to see a difference in the frameworks used. One trivial consequence is that the hashes on the relevant websites do not correspond to MD5 or SHA hashes, and their output cannot be easily identifiable as the output of a common hashing algorithm due to the character set and length parameters. A second visible difference is that websites generally discourage users from using passwords, privileging alternative methods such as unlocking through one's phone, as Google recently deployed on its own service. This means that they also generally implement some forms of 2-factor authentication based on cellphone usage. There are two advantages to this design, in a context where some ISO protocols could potentially be compromised [18,46]. The first is that it makes it easier to prevent foreign actors from being able to decrypt password data exchanged with — potentially compromised — ISO protocols while it is in transit. The second is that, as 2-factor authentication is used, tracking users — through triangulation, among other methods — becomes possible with the cooperation of telephone companies [4]. Strong state security incentives and a tighter cooperation — than in the western world — between the state and large technology companies [45] combined made it feasible to implement on a large (national) scale this kind of technological decision.

The improved security linked to client-based hashing could then be a side-effect of state-wide protection mechanisms against foreign actors. However, the real question is not why those 8 websites implement client-side hashing, but rather, why the others do not implement it.

Server-side hashing in other countries. There are many potential arguments as to why server-side hashing is so frequent, but the main explanation is probably the simplest: inertia and simplicity. In a world where large companies with hundreds of millions of users (such as Mate1) still store their passwords in cleartext, the question is not so much "why is the hashing not done on the client?" but rather "why is the hashing not done at all, or with obsolete tools?", as shown in Table 1. This is compounded by the fact that, unlike the general issue of hashing on which there was a quasi-unanimity and a common push from the security community for more than two decades, the issue of server-side versus client-side hashing is less known, and even academic endeavours didn't question some of the common assumptions until recently [20,35]. Two other issues amplify this inertia and are worth looking into.

The first is that there has been a long tradition of pitting security and functionality against each other. Until recently, common practice said that any improvement on the first came at the expense of the other. This view has recently been challenged, thankfully, as certain designs can in practice improve both [5]

[4] This would be a natural extension of the 2002 law that forced cybercafe owners to keep a list linking login information and state ID for all their clients [44] — in a country where cybercafe was the main internet access point for more than a quarter of users in 2006 [9].

— similarly to how the increased complexity of password constraints in the 2000s actually worsened both security and functionality [29].

The second issue, related to the first, is the incentive structure that surrounds password security. Most online companies operate in an ecosystem where security is not a cost that is paid continuously but instead where they pay nothing until a major leak is made public. As such, there is little in the way of incentives to push those companies to keep up to date against threats they are misinformed about. This translates to developer culture, where security can become an afterthought when the goal is to implement the different functionalities as fast as possible. Even developers aware of the security risks might end up with managerial directives that go against their warnings, as the potential damage can be underestimated until the damage is done [6]. This *reactive* way of handling security is alas poorly adapted to passwords as they have a domino effect on other services [23]. Solving this bad incentive structure — at least on this front — is one of the main advantages of making client-side hashing the norm, as shown in the next section.

4 Cost analysis of client-side hashing

Before we discuss how to implement client-side hashing on a large scale, it's time to summarise its advantages and drawbacks.

4.1 Advantages

No credential reuse attack. The main advantage with client-side hashing is that, as the password never leaves the client machine, database leaks are much less serious. In any case, if an appropriate hashing algorithm and salt are used, an adversary with access to the database cannot reuse the credentials to mount an attack on a different service provider.

Lower server costs. The second advantage is that, as the hashing happens client-side, some server resources are freed, unlike when they have to compute expensive key derivation functions.

Stronger hashing. The previous advantage means that there is no need to compromise between server utilisation and security, as determined by the slowdown factor of the hashing function. A lot of computing power can then be dedicated to hashing, at the client's expense (as they have a low probability of noticing).

Makes phishing more difficult. If the method becomes standardised, the use of the website address as salt can be detected (or corrupt password hashes generated instead). This can help against homograph attacks — where a unicode character that is visually similar is used to get realistic-looking impostor domain names [21] — as one among a set of other mitigation methods [19,36].

Simplicity. As the method can become standardised, and visible in its standardisation, it puts the onus on what happens on the client's side, instead of the server. This leaves more opportunities to improve the database design and the server optimisation, without jeopardising security.

Accountability. The final advantage is that, if implemented at scale, this method can create a social cost for companies that do not implement client-side hashing, as they become known for having lax security practices. In consequence, the cost is transferred to the developers, who have a direct interest in improving the security. This is opposed to what happens currently, as explained earlier, as most developers spend time on security issues only in a reactive manner, after the leak has already happened. This allows the system to have detectable issues that are not only observable through catastrophic failures.

4.2 Drawbacks

There are four central drawbacks to client-side hashing, depending on how it is implemented.

Authentication attacks after leaks. The first issue happens if an attacker manages to obtain a copy of the database. They could then copy the hash and send a valid authentication message to the website. Two factors mitigate this. The first is that it is quite trivial to prevent it by having double hashing, whereby the service provider also runs a minimal amount of hashing server-side, thus preventing this attack. In such a case, the server-side hashing does not require strong security parameters, and a simple SHA-256 is enough [5], as it is not the security bottleneck — as long as the client-side hashing is solid enough to prevent brute-force. The second factor is simply that an adversary able to steal the password database is also most probably able to steal and affect most other systems. As such, the impact would mostly concern buyers of said database rather than the original attacker.

Computing power limits. The second issue is that servers generally have more computing power than at least some of the client devices. As long as most clients authenticate through computers or modern mobile devices, this should not be problematic, as the computing power and, even more importantly, the memory available tend to be more than what many servers could generally afford for a single user, even in an unoptimised Javascript implementation. That said, with the advent of the Internet of Things, some devices with very low power could be involved and require password authentication, which could complicate the matter.

[5] MD5 would not work as it would be easy for an adversary with the leaked database to create an attack: instead of finding the original password, they would only need to find an MD5 collision for it.

Script blocking. A third potential issue — although it is quite minor — is that client-side hashing can be blocked by the client. This is especially true among users who are sensitive to security issues and block all scripts by default. The jump in memory and CPU use could also trigger warnings as they would occur in a way similar to cryptojacking[6] [12].

Incompatibility with legacy protocols. The last issue is quickly disappearing, but is one reason why client-side hashing is still quite rare, through inertia. Some older protocols, especially homebrews, required a cleartext password to function [35].

5 Making changes to the hashing process

As the overwhelming majority of hashing is done server-side today, changing this requires a relatively large amount of labour. We see three main avenues to make the relevant changes, that could all be attempted in parallel.

5.1 A service-centric view

From a service provider point of view, the interest in switching to client-side hashing are akin to those of switching to hashing from initially having stored passwords in cleartext, with a few key differences. The first is that the relative security gains are weaker, whether in terms of real security or in terms of public blame if the security is broken. There is little difference between "adequate" and "strong" security procedures when compared to having "inadequate" security. On the other hand, switching to client-side hashing saves on server costs and code complexity, unlike switching from cleartext to hashed passwords. Hence, although the costs of switching are smaller, the benefits are correspondingly weaker. Moreover, all these are moot points if the incentive structure stays the same, as even the first switch to hashed passwords isn't universal yet.

There is one way to change this incentive structure, by involving major browser developers. A client-side hashing detection system could be integrated into a browser, and give a warning to users when passwords are not handled correctly. This detection system would of course be imperfect and let some websites badly handle passwords while not showing warnings. That said, it could be enough to create a real cost on the service providers, who might lose users to security concerns. Ideally, this could happen in a way similar to what was seen during the switch from HTTP to HTTPS, by first adding warnings and then blocking service providers with unsecure practices (unless the user confirms that they are aware of the risks). Despite the complexity of the architectural changes required [30], browser warnings changed the incentives and had a fast and large-scale impact [40,13]. Finally, convincing one such actor might also probably be

[6] Cryptojacking corresponds to the hidden execution of code inside a browser to mine cryptocurrencies while the user is visiting a website.

enough for the others to follow suit, as other browsers would have some pressure to be perceived as secure to the users as the one displaying the warnings. Adopting some standard header could also help differentiate between websites with probably obsolete security practices and the rest, which would be composed of websites with good security practices and high quality phishing websites [28].

5.2 A user-centric view

From the end user's perspective, the issue is different, as there is a wide variability of possibilities when it comes to users' goals, constraints, and expertise. As long as independent service providers switch to client-side hashing, the process is mostly invisible to users[7], and should have no negative effects. User costs would only appear in one of two cases: if a browser starts implementing warning systems, interrupting users' actions, or if a user decides to take matters in their own hands by using an extension that implements a warning system. We'll start by looking at how the second could happen.

An extension to warn users. The first and easiest short term solution is to create a short script — a priori in the form of a browser extension — to detect whether the password is sent in cleartext to the service provider. This script could be based on some of the methods mentioned earlier, such as detection of the password string in the outgoing packets, or use of computing resources. It could be displayed next to the padlock corresponding to HTTPS connections, in the form of a warning in the address bar — or potentially even more aggressively as a pop-up. The effects on the user would be partially detrimental as it would distract from their current task, although it could help some users avoid using passwords on unsecure websites. The main advantage of this would however be the incentive structure it would create to switch systems if widely deployed.

There is one potential drawback of this method in the form of a privacy risk similar to the one we just started observing on HTTPS padlocks [39,10]. If the warning system shows not only indications of risky websites but also of safe ones, corrupting the warning system itself becomes a worthy goal. As such, users could be more easily fooled by phishing attempts that manage to show good password security than they would with neither positive nor negative warnings. That might be less of an issue because, unlike HTTPS, warning systems for client-side hashing would easily detect bad practice but struggle to detect truly good practice[8], but still bears keeping in mind.

[7] The only way for it to be visible is if it unduly increases delays by asking too many rounds of hashing on a low-powered device, but this is a matter of parameter optimisation where wide margins could be taken by default to avoid this issue.

[8] For example, to be sure the password is not sent in cleartext, one would need to make sure that the password field is accessed exactly once as input to the hash function, otherwise any reversible function could be used before transmitting, dodging accusations of cleartext sending. Similarly, the website could trigger some expensive computation without using it to fool resource monitors.

Detecting and hashing passwords on the client. A more extreme case for more technically inclined — and concerned — users would be to use a different kind of extension, as a stopgap measure. Instead of checking whether the password is sent in cleartext, it would be possible to automatically detect password fields — as Google Chrome does — and offer a second field through the extension. After the user types their password in that second field, the hashed result could be directly input into the original field. This bypasses a few issues and adds some level of security, but would also be harder to optimise than if done natively by the service provider. One concern then would be that the user's password could not be directly used on a different device without the extension. The website changing its domain name would also create problems that are harder to address from this client-centric view.

6 Discussion

We have shown that client-side hashing benefits from multiple advantages, and that its drawbacks often come from older constraints and are quickly becoming less relevant. Despite this, among the most used websites, it is only used today by Chinese service providers, as part of a larger security suite common to many of them. After observing the issues caused by server-side hashing, we provide some ideas to detect such hashing techniques at a larger scale than what we manually did. We also propose integrating them into common browsers to change the incentive structure for developers and companies involved in the security ecosystem. We finish by offering some alternatives for end users, such that all solutions mentioned could be used in parallel.

The changes we propose are minimal and have some self-perpetuating mechanisms, exactly because expecting a sudden and non-trivial change from a large security ecosystem would be idealistic. There are of course alternatives to the solutions proposed, such as Time-based One-time Password algorithms [38], which solve many issues mentioned. The problem, as with all other security improvements, is getting large actors to make the requisite changes. A different alternative is to use password managers — which the hashing extension we mention imitates in some ways — but this brings us back to older security models by shifting all costs to the user. Moreover, password managers still have low penetration on mobile devices and are not always compatible with all users' constraints [3].

We see two ways to go further in the direction we explored. First, it seems wise to investigate whether the increasing role played by low-power devices in the Internet of Things could create bottlenecks security-wise. Second, to increase the amount of hashing time available, one could hash the password letter by letter, using the lapse between keystrokes to hash what is available for a set duration and using this as a salt for the next hash. This is not currently done, and could potentially create security vulnerabilities, so a thorough cryptanalysis of this method should be done with the currently used password hashing functions. On the usability side, there is also the question of finding an ideal delay to resist parallelised attacks without creating a time cost for users on lower-end devices.

Acknowledgements

We're grateful to participants of the Privacy and Security Workshop, IU Gateway Berlin, for their comments. This work was supported partly by the french PIA project "Lorraine Université d'Excellence", reference ANR-15-IDEX-04-LUE.

References

1. Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M.L., Stransky, C.: How internet resources might be helping you develop faster but less securely. IEEE Security Privacy **15**(2), 50–60 (March 2017). https://doi.org/10.1109/MSP.2017.24
2. Acar, Y., Fahl, S., Mazurek, M.L.: You are not your developer, either: A research agenda for usable security and privacy research beyond end users. In: IEEE Cybersecurity Development – SecDev. pp. 3–8 (Nov 2016). https://doi.org/10.1109/SecDev.2016.013
3. Alkaldi, N., Renaud, K.: Why do people adopt, or reject, smartphone password managers? In: Proceedings of EuroUSEC (2016), eprint on Enlighten: Publications
4. Amazon Alexa: 500 global sites (2019), `http://alexa.com/topsites/`
5. Baskerville, R., Rowe, F., Wolff, F.C.: Functionality vs. security in is: Tradeoff or equilibrium. In: International Conference on Information Systems. pp. 1210–1229 (2012)
6. Baskerville, R., Spagnoletti, P., Kim, J.: Incident-centered information security: Managing a strategic balance between prevention and response. Information & management **51**(1), 138–151 (2014)
7. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy. pp. 72–84. IEEE (1992)
8. Biryukov, A., Dinu, D., Khovratovich, D.: Argon2: new generation of memory-hard functions for password hashing and other applications. In: IEEE European Symposium on Security and Privacy – EuroS&P. pp. 292–302. IEEE (2016)
9. Center, C.I.N.I.: 18th statistical survey report on the internet development in china. Tech. rep., CINIC (2006)
10. Cimpanu, C.: Extended validation (ev) certificates abused to create insanely believable phishing sites (2017), `https://www.bleepingcomputer.com/news/security/extended-validation-ev-certificates-abused-to-create-insanely-believable-phishing-sites/`
11. Dürmuth, M., Kranz, T.: On password guessing with gpus and fpgas. In: Mjølsnes, S.F. (ed.) Technology and Practice of Passwords. pp. 19–38. Springer International Publishing, Cham (2015)
12. Eskandari, S., Leoutsarakos, A., Mursch, T., Clark, J.: A first look at browser-based cryptojacking. In: 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 58–66. IEEE (2018)
13. Felt, A.P., Barnes, R., King, A., Palmer, C., Bentzel, C., Tabriz, P.: Measuring {HTTPS} adoption on the web. In: 26th {USENIX} Security Symposium ({USENIX} Security 17). pp. 1323–1338 (2017)
14. Florêncio, D., Herley, C., van Oorschot, P.C.: An administrator's guide to internet password research. In: LISA. vol. 14, pp. 35–52 (2014)
15. Ge, C., Xu, L., Qiu, W., Huang, Z., Guo, J., Liu, G., Gong, Z.: Optimized password recovery for sha-512 on gpus. In: IEEE International Conference on Computational Science and Engineering – CSE – and Embedded and Ubiquitous Computing – EUC. vol. 2, pp. 226–229. IEEE (2017)

16. Goodin, D.: Once seen as bulletproof, 11 million+ ashley madison passwords already cracked (2015), https://web.archive.org/web/20180803014106/https://arstechnica.com/information-technology/2015/09/once-seen-as-bulletproof-11-million-ashley-madison-passwords-already-cracked/
17. Green, M.: Let's talk about pake (2018), https://web.archive.org/web/20190426024348/https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/
18. Hales, T.C.: The NSA back door to NIST. Notices of the AMS **61**(2), 190–19 (2013)
19. Hannay, P., Baatard, G.: The 2011 idn homograph attack mitigation survey. In: Proceedings of the International Conference on Security and Management (SAM'12) (2012)
20. Hatzivasilis, G., Papaefstathiou, I., Manifavas, C.: Password hashing competition-survey and benchmark. IACR Cryptology ePrint Archive (2015)
21. Holgers, T., Watson, D.E., Gribble, S.D.: Cutting through the confusion: A measurement study of homograph attacks. In: USENIX Annual Technical Conference, General Track. pp. 261–266 (2006)
22. Independent Security Evaluators: Password managers: Under the hood of secrets management. Tech. rep., ISE (2019), https://web.archive.org/web/20190301171335/https://www.securityevaluators.com/casestudies/password-manager-hacking/
23. Ives, B., Walsh, K.R., Schneider, H.: The domino effect of password reuse. Communications of the ACM **47**(4), 75–78 (Apr 2004). https://doi.org/10.1145/975817.975820
24. Jaeger, D., Pelchen, C., Graupner, H., Cheng, F., Meinel, C.: Analysis of publicly leaked credentials and the long story of password (re-) use. In: Proc. Int. Conf. Passwords (2016)
25. Jarecki, S., Krawczyk, H., Xu, J.: Opaque: an asymmetric pake protocol secure against pre-computation attacks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 456–486. Springer (2018)
26. Karyda, M., Mitrou, L.: Data breach notification: Issues and challenges for security management. In: Mediterranean Conference on Information Systems (2016)
27. Khandelwal, S.: Facebook caught asking some users passwords for their email accounts (2019), https://web.archive.org/web/20190404071339/https://amp.thehackernews.com/thn/2019/04/facebook-email-password.html
28. Kisa, K., Tatli, E.: Analysis of http security headers in turkey. International Journal of Information Security Science **5**(4), 96–105 (2016)
29. Komanduri, S., Shay, R., Kelley, P.G., Mazurek, M.L., Bauer, L., Christin, N., Cranor, L.F., Egelman, S.: Of passwords and people: Measuring the effect of password-composition policies. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2595–2604. CHI '11, ACM, New York, NY, USA (2011). https://doi.org/10.1145/1978942.1979321
30. Kranch, M., Bonneau, J.: Upgrading https in mid-air. In: Proceedings of thz 2015 Network and Distributed System Security Symposium. NDSS (2015)
31. Krebs, B.: Twitter to all users: Change your password now! (2018), https://web.archive.org/web/20190402093127/https://krebsonsecurity.com/2018/05/twitter-to-all-users-change-your-password-now/
32. Krebs, B.: Facebook stored hundreds of millions of user passwords in plain text for years (2019), https://web.archive.org/web/20190322091235/https://krebsonsecurity.com/2019/03/facebook-stored-hundreds-of-millions-of-user-passwords-in-plain-text-for-years/

33. Kumar, H., Kumar, S., Joseph, R., Kumar, D., Singh, S.K.S., Kumar, P.: Rainbow table to crack password using md5 hashing algorithm. In: IEEE Conference on Information & Communication Technologies – ICT. pp. 433–439. IEEE (2013)
34. MartinKauppi, L.B., He, Q.: Performance Evaluation and Comparison of Standard Cryptographic Algorithms and Chinese Cryptographic Algorithms. Master's thesis (2019)
35. Mazurek, M.L., Komanduri, S., Vidas, T., Bauer, L., Christin, N., Cranor, L.F., Kelley, P.G., Shay, R., Ur, B.: Measuring password guessability for an entire university. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security. pp. 173–186. CCS '13, ACM, New York, NY, USA (2013). https://doi.org/10.1145/2508859.2516726
36. McElroy, T., Hannay, P., Baatard, G.: The 2017 idn homograph attack mitigation survey. In: Proceedings of the 15th Australian Information Security Management Conference (2017)
37. Morris, R., Thompson, K.: Password security: A case history. Communications of the ACM **22**(11), 594–597 (Nov 1979). https://doi.org/10.1145/359168.359172
38. M'Raihi, D., Machani, S., Pei, M., Rydell, J.: Rfc6238: Totp: Time-based one-time password algorithm (2011), https://tools.ietf.org/html/rfc6238
39. Peng, P., Xu, C., Quinn, L., Hu, H., Viswanath, B., Wang, G.: What happens after you leak your password: Understanding credential sharing on phishing sites. In: AsiaCCS 2019. pp. 181–192 (07 2019). https://doi.org/10.1145/3321705.3329818
40. Schechter, E.: Moving towards a more secure web (2016), https://web.archive.org/web/20190405120627/https://security.googleblog.com/2016/09/moving-towards-more-secure-web.html
41. Shape: 2018 credential spill report. Tech. rep., Shape Security (2018)
42. Siegrist, J.: Lastpass hacked – identified early & resolved (2015), https://web.archive.org/web/20190412054716/https://blog.lastpass.com/2015/06/lastpass-security-notice.html/
43. Sprengers, M.: GPU-based Password Cracking. Master's thesis, Radboud University Nijmegen (2011)
44. State Council of the People's Republic of China: Regulations on administration of business premises for internet access services, article 23 (2002)
45. Swaine, M.D.: Chinese views on cybersecurity in foreign relations. China Leadership Monitor (42) (2013)
46. Tryfonas, T., Carter, M., Crick, T., Andriotis, P.: Mass surveillance in cyberspace and the lost art of keeping a secret. In: International Conference on Human Aspects of Information Security, Privacy, and Trust. pp. 174–185. Springer (2016)
47. Vyas, T., Dolanjski, P.: Communicating the dangers of non-secure http (2017), https://web.archive.org/web/20190524003142/https://blog.mozilla.org/security/2017/01/20/communicating-the-dangers-of-non-secure-http/
48. Whittaker, Z.: Github says bug exposed some plaintext passwords (2018), https://web.archive.org/web/20190331110732/https://www.zdnet.com/article/github-says-bug-exposed-account-passwords/
49. Wiemer, F., Zimmermann, R.: High-speed implementation of bcrypt password search using special-purpose hardware. In: International Conference on ReConFigurable Computing and FPGAs – ReConFig. pp. 1–6. IEEE (2014)
50. Wu, T.: The SRP authentication and key exchange system. Tech. rep., RFC Editor (2000)