Explicit Delegation using Configurable Cookies

David Llewellyn-Jones, Graeme Jenkinson, and Frank Stajano

Computer Laboratory, University of Cambridge, Cambridge, UK,
{david.llewellyn-jones, graeme.jenkinson, frank.stajano}@cl.cam.ac.uk,
https://mypico.org

Abstract. Password sharing is widely used as a means of delegating access, but it is open to abuse and relies heavily on trust in the person being delegated to. We present a protocol for delegating access to websites as a natural extension to the Pico protocol. Through this we explore the potential characteristics of delegation mechanisms and how they interact. We conclude that security for the delegator against misbehaviour of the delegatee can only be achieved with the cooperation of the entity offering the service being delegated. To achieve this in our protocol we propose configurable cookies that capture delegated permissions.

1 Introduction

Everybody hates passwords. They are insecure in theory and even worse in practice; complex passwords are slow to use and prone to error. The Pico Project[1] [10] aims to provide a secure, privacy-protecting hardware token replacing all of a user's passwords while avoiding the need for trusted third parties and eliminating phishing attacks. To be successful, Pico must give not just the same level of usability, but also the same level of flexibility as passwords. It must do this even where the functionality is currently achieved by subverting security mechanisms to such an extent that the security benefits of passwords are essentially lost. This is quite a tall order.

A case in point is that of *delegation*. For most websites there is no formal approach for delegating access to an account, but users still successfully manage to do so on a quite routine basis. How do they do this? They share their password with the person they're delegating to.

It will be immediately obvious that this provides little-to-no security above that offered by the delegator's trust in the person being delegated to. This delegatee is granted full permissions to the account, to the extent they could quite easily lock the rightful owner out by changing the password.

Users aren't doing this because they lack a basic security understanding. They do it because they trust the people involved or societal pressures, because it's convenient, because they have no alternative, and because it works [9]. Sharing of passwords is even used as an expression of trust [7,8].

The challenge is to provide a more secure approach that users will be willing to adopt. We ask the question of how Pico, building on top of the incumbent

[1] https://mypico.org

Authors' preprint of 2016-06-13.
Final version to appear in *Proceedings of Security Protocols Workshop 2016*, Springer LNCS.

infrastructures of the Web, can offer similarly simple delegation but without destroying its security benefits in the process. We will explore this question, showing that a simple delegation protocol is not only possible, but is in fact a natural evolution of the existing Pico design. We claim the following contributions in this paper.

1. A technical solution for delegation offering security, minimal reliance on the goodwill of the delegatee and using Web technologies for easy deployment (section 2.1).
2. A taxonomy to explore the space of possible delegation solutions (section 3).
3. A claim that you cannot delegate securely and revocably without introducing technical changes to the verifier (section 3.1).
4. A discussion about open questions, including how best to deploy evolvable solutions: better to provide a formal mechanism early that relies on trust, or a more secure solution requiring changes to website back-ends (section 4)?

2 Pico the Delegation Device

In its broadest terms delegation is an arrangement between three parties: the *delegator* user, the *delegatee* user and the *delegated* service. We will refer to them respectively as *Rebecca* and *Eric*, accessing the *DumpIt* cloud storage service. The semantic relationship between the parties involved is shown in Figure 1.

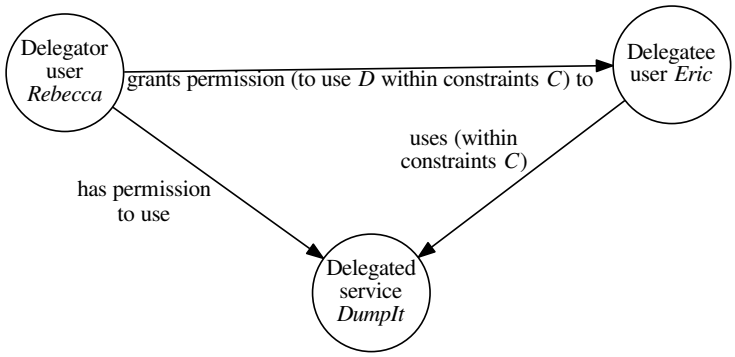


Fig. 1: Semantic relationship between the actors.

A critical observation for us was that, in its existing form, Pico is already performing delegation according to the relationship as shown. To see this, consider the standard Pico protocol for logging in to an account on the Web shown in Figure 2 and which is documented in more detail by Jenkinson *et al.* [6]. The process is initiated by Rebecca entering a URL on her browser with the Pico Lens installed. On receiving the login page, the Lens displays a QR code then

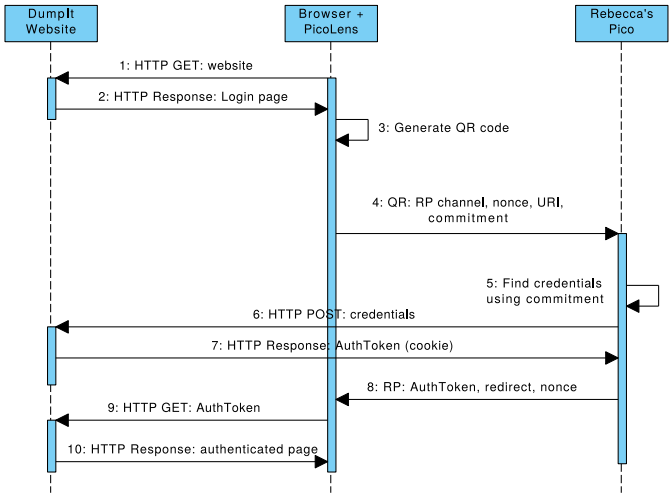


Fig. 2: The standard Pico protocol.

scanned by the Pico. This provides the Pico with the necessary information to log in to the DumpIt site.

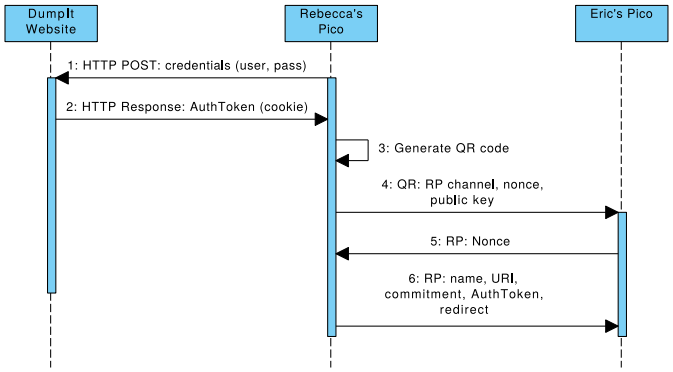
At this point the real authentication begins. The authentication takes place between DumpIt and Rebecca's Pico, with the end result that Rebecca's Pico has the digital *AuthToken* (a session cookie) for accessing the site. But even though Rebecca's Pico has authenticated, this is of no use to Rebecca. She doesn't want to access the site using her Pico, she wants to access it on a more convenient device with a decent browser. Hence Rebecca's Pico *delegates* the AuthToken to the browser in step 8 of the protocol.

So Pico is already a delegation device, but with a pool of delegatees restricted to lenses it's already paired with. We can generalise this to allow delegation to other humans by breaking step 8 into two and placing another Pico in the middle.

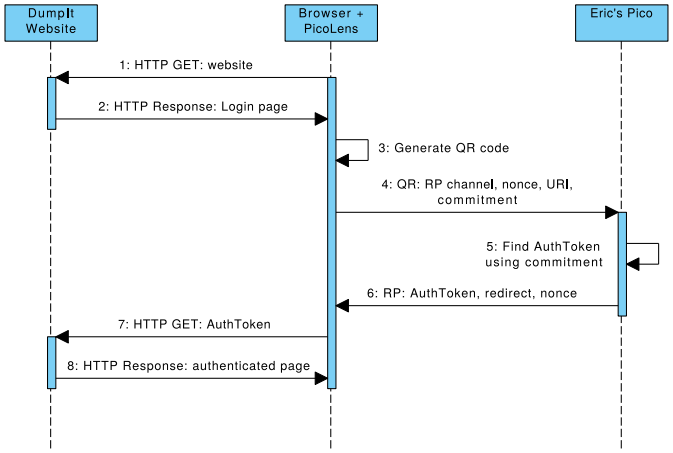
2.1 Delegation Protocol Using Cookies

The resulting split protocol is shown in Figures 3a and 3b. This allows an authorisation token (the session cookie) to be delegated from Rebecca's Pico – which has an account with the DumpIt service – to Eric's Pico, allowing Eric to access DumpIt as if he were Rebecca (and even if Eric has no account with DumpIt himself).

The process involves Rebecca passing the AuthToken to Eric that he's then able to present to DumpIt for access to the service. As can be seen in Figure 4a, the actual delegation step is initiated when Rebecca selects the option on her Pico. Her Pico displays a QR code which Eric scans, instigates the transfer of the AuthToken from Rebeca's Pico to Eric's Pico where it can be stored for later use. The use of a transient QR code is intended to encourage Eric to be in physical proximity with Rebecca when the transfer takes place, although in



(a) Delegating an AuthToken.

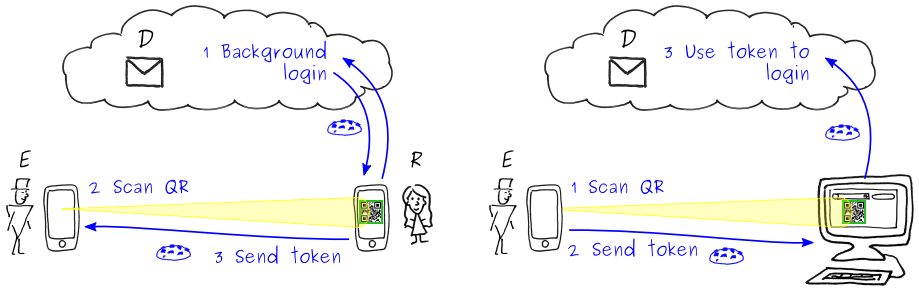


(b) Using a delegated AuthToken.

Fig. 3: The delegation protocol, slightly simplified for clarity.

and of itself it can't guarantee this. In the background Rebecca is `POST`-ing her credentials to DumpIt, which returns back a session cookie. This session cookie is the AuthToken passed to Eric.

Some time later Eric will want to make use of the delegated token. Eric can make use of the existing Pico approach for logging in to websites: he scans a QR code presented on DumpIt's login page, providing the Pico with the information needed to complete the login. As we've seen the usual Pico approach would have Eric's Pico `POST` login credentials (username and password) to the website, which would reply back with a session cookie for access to the site. The Pico then installs this on the browser. However, the AuthToken held by Eric's Pico is already such a session cookie. Rather than `POST` credentials, Eric's Pico sends the AuthToken to the browser, which uses it for access to the site without the need to log in. Figure 4 illustrates how the process appears to the end user.



(a) Delegating an AuthToken. (b) Using a delegated AuthToken.

Fig. 4: Delegation in practice.

We have implemented this on top of the existing Android Pico App and Pico Lens Firefox Plugin. This has provided us some insight into how effective the approach is, and its prospects for providing improvements in the future. It has also raised important questions about whether users would make use of the functionality even if Pico made it available. Is delegation a practical function?

3 Delegation for End Users

To answer this question we believe the social characteristics of delegation are key. The goal of delegation typically falls in to one of five categories as shown in Figure 5. We offer an example of each.

	Shareable	*Exclusive*	
Grant access	Effort-sharing	Effort-transfer	*Task*
	Value-sharing	Value-transfer	*Benefit*
Restrict access	Self-delegation		

Fig. 5: The five categories of delegation.

Value-sharing involves multiple users all gaining non-exclusive benefit from a service through delegation of access. Netflix allows the legitimate sharing of an account for simultaneous streaming (ideal for adults sharing a flat). It supports multiple profiles but credentials still have to be shared to allow access to content. Sharing the credentials for simultaneous access multiplies the value of the account to its users.

Effort-sharing is similar, but now multiple users are able to share a task amongst them through access to a service. A reviewer assigned multiple papers to review through a conference website might delegate some of the reviews to a colleague by sharing access[2]. A similar example is that of a corporate social media account[3]. The communications team will need access to a single company account, with the only way of doing so by sharing the credentials between them. The effort of running the media feed is divided between all of those involved.

Effort-transfer involves the delegation of a specific task from one person to another and is similar to effort-sharing, but with the addition of exclusivity. If I want my local computer-shop to health-check my laptop I'll have to hand over the admin password. We can't both work on it simultaneously and ideally I'd like to give them access only while they're performing the task on my behalf. I'll have to give them admin access too, even though they don't need to see my photos and emails.

Value-transfer shifts the process back from activity to consumption again. Your account credentials represent the key to accessing some value you may want to 'gift' to someone else. For example in the past parents would hand over cash to their children to buy sweets from the local shop. Now they send them out with a credit card and PIN [9]. Either way, once the money is spent it can't be used again.

Clearly there can be some blurring of boundaries between these categories. For example a social media account will require effort to maintain, but allows users to draw benefit from it at the same time.

Self-delegation reverses the role of delegation. Rather than granting access to others it can be used to *restrict* access for the owner of the account; useful for managing risk. For example while travelling abroad you might want to take with you access to only low-value and essential accounts. You can delegate just the accounts you need, limiting your exposure in case they are compromised. This is similar to buying a pre-paid cash card for travel.

One of these categories is not like the others. The first four are all *permissive*, whereas the last is *restrictive*. This is also reflected in how they can be performed. The first four can all be achieved using passwords, but self-delegation cannot.

For the permissive categories, password sharing is likely to be a simple way to achieve the objective. It is unlikely to be the safest or most secure way of doing so, but with an appropriate level of trust most people will accept this (essentially replacing technical security with personal trust).

It is worth noting that there is another form of delegation that implies an obligation on the party delegated to, as happens when a task is delegated from one person to another [5]. In spite of the terminology we use (*e.g. effort-sharing* and *effort-transfer*) we're only interested in technical enforcement related to delegation of permissions, not tasks. Although the two may go together (you

[2] In practice, most peer-review platforms provide a formal mechanism for delegating reviews, so sharing an account wouldn't be necessary.

[3] For example Twitter or Instagram.

Variable	Type	Values
Usability	continuous	low – high
Expressiveness/flexibility	continuous	low – high
Security	continuous	low – high
Trust required	continuous	low – high (of E by R)
Accountability of E	discrete	$\emptyset$, E, R, D, ER, ED, RD, all; *i.e.* $\mathcal{P}(\{E, R, D\})$
Plausible deniability of E	discrete	full, RD, ED, ER, D, R, E, $\emptyset$
Involvement at creation	discrete	$\widehat{ER}$, ER, ED, $\widehat{ERD}$, ERD
Involvement at operation	discrete	ED, EDR
Revocable by	discrete	$\emptyset$, R, D, RD

Table 1: Delegation scope. A spanning arc $\widehat{XY}$ implies a physical proximity constraint, described further in Section 3.1.

may have to delegate some permissions in order for a task to be completed) we won't consider how such task obligations might be enforced.

Delegation, then, is a rich space in terms of user motivation and functionality. In creating an effective delegation process we must encompass this functionality with our technical protocol, but the design choices we make in achieving this will also affect other characteristics which are also important, such as usability and security. Mapping approaches to points in the space of relevant characteristics can provide us some insight into compromises that might incentivise users to choose one approach (password sharing?) over another (Pico delegation?).

3.1 Constraints

By definition the process of delegation enforces certain constraints in what any protocol must involve, such as the fact that Rebecca must be involved in the set up of the delegation step, and both Eric and DumpIt must be involved in the use of the delegated privilege.

However there remain several areas that are open to interpretation. Table 1 presents our attempt to define the possible variables that can be used to categorise different delegation approaches. The table summarises the variables but requires some explanation in places.

Expressiveness relates to the use of a language to specify the permissions granted (contrariwise, specifying limits to their use). For example, Rebecca may want to delegate only a subset of the available functionality of DumpIt, to allow reading but not writing, or to apply a time restriction on how long the service can be accessed.

The trust required implies the extent to which Rebecca must trust Eric not to abuse the delegated permissions. Allowing more precise permissions specification and support for revocation will reduce the level of trust needed.

The values for accountability list those actors with the potential to account for Eric's actions in a provable way. Plausible deniability represents the inverse

of this. It indicates whether Eric can deny responsibility for an action to a particular set of other entities.

Whether a party is involved at the creation determines how the process of delegation takes place (by definition delegation always occurs from Rebecca to Eric). A spanning arc $\widehat{ER}$ implies Eric and Rebecca must be in physical proximity to perform the delegation (otherwise conducted remotely via the network). Both the delegatee and delegated service must always be involved when the permission is used ("involved at operation"). Requiring the delegator to be involved implies that every action can be permitted or refused in real time.

Finally, revocation implies that Eric's permissions may be withdrawn without affecting those of Rebecca.

Ideally a developer could simply select the best for each variable and forge a new protocol to suit. Unfortunately there are also constraints between them to prevent them being selected *à la carte*. We capture these in the following claim.

Claim 1. The following relationships are intrinsic to the nature of delegation.

1. There is an inverse relationship between expressiveness (supporting a richer language for expressing rights delegation) and usability.
2. There is a direct relationship between expressiveness and security.
3. There is an inverse relationship between security and trust.
4. Accountability is the precise inverse of plausible deniability. Therefore if Eric can plausibly deny some act, then he cannot be accountable for it.
5. Revocation implies accountability. If a party can revoke Eric's permissions without revoking Rebecca's, then that party must be able to tell which of the two is performing an operation.

A number of further relationships would seem to immediately derive from these, such as that there is an inverse relationship between usability and security. For discussion and examples relating trust, security, delegation, plausible deniability and authentication together, see Christianson's insightful paper [4]. We also claim the relationship between security, trust and expressiveness to be of crucial importance, which we express as follows.

Claim 2. For Rebecca to minimise her exposure she must restrict the permissions entrusted to Eric to just those needed for him to carry out his task.

A more expressive rights language will allow permissions to be expressed more specifically and therefore reduce trust. Since the permissions are dependent on the functionality offered by the service, and the gate to this functionality must also be provided by the service, this immediately leads us to our final claim.

Claim 3. The security-trust balance of delegation can only be achieved if the verifier offers a means of expressing fine-grained permissions.

We'll return to this in section 3.3 where we discuss the practical implications of this in relation to cookies. Since the baseline case in terms of both functionality and usability is that provided by password sharing, we will consider how this fits within these constraints in the next section.

3.2 Characteristics using Passwords as Tokens

As we've seen, delegation is already perfectly possible just by sharing passwords. In their seminal paper of 1999, Adams and Sasse recommend shared passwords as an important way for authentication to match workers' sense of responsibility: "Shared work and responsibility require users to perceive that they are using shared passwords" [1]. They also identify that it can impact security and more recent advice from GCHQ emphasises this unequivocally: "You should never allow password sharing between users. Sharing accounts, or even occasional use by anyone other than the account holder, negates the benefit of authenticating a specific user." [3].

In practice, for most websites, sharing of passwords is the only practical way for an end user to delegate access. If our claims in the previous section concerning the constraints on delegation are valid, we should be able to place password sharing into the space and demonstrate how a more formal approach could move it to a 'better' place. We use the variables from Table 1 to summarise the characteristics as follows.

$$\begin{aligned} \mathit{usability} &= \text{high}, & \mathit{flexibility} &= \min, & \mathit{security} &= \min, \\ \mathit{trust} &= \max, & \mathit{accountability} &= \emptyset, & \mathit{deniability} &= \text{full}, \\ \mathit{creation} &= \widehat{ER}, & \mathit{operation} &= ED, & \mathit{revocable} &= \emptyset. \end{aligned}$$

Our underlying claim is that without a formally defined or approved way to perform delegation, users will continue to simply share passwords as a means to achieve it. And yet password sharing is inherently insecure (as indicated by the *security* and *trust* assignments). For delegation to evolve positively, we must supply sensible protocols for performing delegation that increase *security* and decrease *trust* while having minimal negative impact on *usability*.

In the next section we will consider our proposed protocol using cookies as tokens and discuss how it provides a framework that can evolve to support this.

3.3 Characteristics using Cookies

The characteristics depend heavily on the contents of the AuthToken. To take a common example, a WordPress admin cookie contains the following fields[4].

$$\begin{aligned} \mathit{cookie} &= \mathit{username} \mid \mathit{expiration} \mid \mathit{token} \mid \mathit{hash}. \\ \mathit{hash} &= \mathit{SHA256}(\mathit{username} \mid \mathit{expiration} \mid \mathit{token}, \\ &\qquad \mathit{MD5}(\mathit{username} \mid \mathit{passfrag} \mid \mathit{expiration} \mid \mathit{token})). \end{aligned}$$

In all cases *expiration* is the same ASCII string representing the time in Unix epoch format and *token* is a random session token stored against the user in

[4] See `https://developer.wordpress.org/reference/functions/wp_generate_auth_cookie/`.

Scheme	Usability	Flexibility	Security	Trust	Accountability	Deniability	Creation	Operation	Revocable
Passwords	High	Min	Min	Max	$\emptyset$	Full	$\widehat{ER}$	ED	$\emptyset$
Cookies	Med	Med	Med	Med	RD	R, D	$\widehat{ER}$	ED	D

Table 2: Property comparison.

the WordPress database. The *passfrag* string is a four character fragment of the base64-encoded hashed password.

The key points to note are that the data – including the expiration time – are all HMAC-ed and that the cookie includes a per-session random token. Consequently there's no way for Eric to extend the expiration time without invalidating the cookie, and DumpIt can revoke the cookie by disassociating its database entry for the session token from the user. Finally, although DumpIt isn't able to distinguish between normal and delegated sessions, Rebecca can do so since every AuthToken (each with unique session token field) has to pass through her Pico. If DumpIt were to record activity against sessions it could, in collaboration with Rebecca, account for Eric's activity.

Under this scheme we therefore have the following properties for cookies, which are also summarised in Table 2 for easy comparison against passwords.

$$\begin{array}{rrr} \mathit{usability} = \text{medium}, & \mathit{flexibility} = \text{medium}, & \mathit{security} = \text{medium}, \\ \mathit{trust} = \text{medium}, & \mathit{accountability} = RD, & \mathit{deniability} = R, D \\ \mathit{creation} = \widehat{ER}, & \mathit{operation} = ED, & \mathit{revocable} = D. \end{array}$$

This can all be achieved by leveraging the existing site's cookie mechanisms. WordPress offers a particularly robust cookie structure; other sites are less thorough, for example relying on the default HTTP cookie expiration fields which are not cryptographically tied to the cookie (and can therefore be changed).

The only flexibility WordPress offers when generating cookies is whether they last for two days or two weeks[5]. This would allow Rebecca some control over how long Eric is granted access, but doesn't offer more expressive control over the admin capabilities delegated to him. Nevertheless this is an improvement on password delegation. As such we describe this property as *medium*, although a different cookie structure could allow greater flexibility (*low* to *high*).

[5] These values are hardcoded and chosen using the `rememberme` form value; see `https://developer.wordpress.org/reference/functions/wp_set_auth_cookie/`.

This is an example of our Claim 3 above. The security is limited by the expressiveness of the permissions supported by the verifier. In our Pico implementation the website takes on the role of verifier, and so the website must be updated to support fine-grained permission and to allow Rebecca to bake configurable cookies expressing them to her specification. In theory the website could choose any rule language for this, but in practice Rebecca specifies the rules using an interface on her Pico, so some agreed interaction is required.

4 Open Questions

Question 1. Until websites support configurable cookies with more fine-grained permissions, the Pico implementation can still offer all-or-nothing delegation. Given the prevalence of password reuse across sites, this represents an improvement on password sharing. Is it therefore appropriate to make this functionality available even in this sub-optimal form, or better to wait until sites offer the configurable cookies required for proper security?

Question 2. Our implementation using Pico offers some insight into what is achievable. But would real people actually use the functionality? A user study by Bauer *et al.* [2] found people "are most likely to pick the [delegation] option they understand the best, even when they know it is not the option they want.". Will users take the time to learn about and apply tight delegation rules?

Question 3. In the form of password sharing, delegation is an expression of trust [8]. If I give you my password I'm not just granting you access, I'm also making a statement. Is there a way to avoid formal delegation processes becoming a mark of distrust?

Question 4. Delegation is transitive in our implementation. Therefore an AuthToken granted to one user is easily transferred to another. The delegatee doesn't even need an account with the delegated service. Can we enforce non-transitive delegation while still supporting this, and without conflicting with plausible deniability?

Question 5. Many sites offer REST-ful APIs intended for delegated access by trusted apps. Developer tokens or delegated authorisation (*e.g.* OAuth) are often used to allow access. Do these provide a better starting point than cookies for user delegation?

5 Conclusion

Password sharing is widely used as a means of delegating access, so providing a more formal method seems like an obvious next step. We have described an approach that is a natural extension to the existing Pico protocol. We found the most important lever for controlling security to be the expressiveness of the language used to describe permissions delegated, and that configurable cookies are a means for achieving this. Exploring this space has uncovered many questions that still need to be answered.

Acknowledgements

We are grateful to the European Research Council for funding this research through grant StG 307224 (Pico). We also thank the workshop attendees for comments.

References

1. Adams, A., Sasse, M.A.: Users are not the enemy. Commun. ACM 42(12), 40–46 (Dec 1999), http://doi.acm.org/10.1145/322796.322806
2. Bauer, L., Cranor, L.F., Reiter, M.K., Vaniea, K.: Lessons learned from the deployment of a smartphone-based access-control system. In: Proceedings of the 3rd Symposium on Usable Privacy and Security. pp. 64–75. SOUPS 07, ACM (2007)
3. CESG: Password Guidance: Simplifying Your Approach. CESG, CPNI (Jan 2016), https://www.cesg.gov.uk/guidance/password-guidance-simplifying-your-approach
4. Christianson, B.: Living in an impossible world: Real-izing the consequences of intransitive trust. Philosophy & Technology 26(4), 411–429 (Aug 2013)
5. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Modeling security requirements through ownership, permission and delegation, pp. 167–176. IEEE (Aug 2005)
6. Jenkinson, G., Spencer, M., Warrington, C., Stajano, F.: I Bought a New Security Token and All I Got Was This Lousy PhishRelay Attacks on Visual Code Authentication Schemes, pp. 197–215. Lecture Notes in Computer Science, Springer International Publishing (Mar 2014), http://link.springer.com/chapter/10.1007/978-3-319-12400-1_19
7. Lenhart, A., Lewis, O., Rainie, L.: Teenage life online: The rise of the instant-message generation and the internets impact on friendships and family relationships (Jun 2001), http://www.pewinternet.org/2001/06/21/teenage-life-online/
8. Palfrey, J., Sacco, D.T., danah boyd, DeBonis, L., Tatlock, J.: Enhancing Child Safety and Online Technologies (Dec 2008), http://cyber.law.harvard.edu/pubrelease/isttf/
9. Singh, S., Cabraal, A., Demosthenous, C., Astbrink, G., Furlong, M.: Password Sharing: Implications for Security Design Based on Social Practice, p. 895904. CHI 07, ACM (2007), http://doi.acm.org/10.1145/1240624.1240759
10. Stajano, F.: Pico: no more passwords! In: Proceedings of the 19th international conference on Security Protocols. pp. 49–81. SP'11, Springer-Verlag, Berlin, Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-25867-1_6