2010-08-12

BackFlip: A Principled Approach to Online Attribute Verification

Devlin R. Daley
Brigham Young University - Provo

BackFlip: A Principled Approach to Online Attribute Verification

Devlin Daley

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Phillip J. Windley, Chair
Kent E. Seamons
Eric G. Mercer

Department of Computer Science

Brigham Young University

December 2010

ABSTRACT

BackFlip: A Principled Approach to Online Attribute Verification

Devlin Daley

Department of Computer Science

Master of Science

As traditional interactions in the real-word move online, services that require verified personal information from web users will increase. We propose an architecture for the verification of web user attributes without the use of cryptographic-based credentials. In this architecture, service providers are delegated a user's ability to directly contact a certifying party and retrieve attribute data. We demonstrate that this approach is simple for both developers and users, can be applied to existing Internet facilities and sufficiently secure for typical web use cases.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Phil Windley. I have appreciated his patience, his interest in me and my education, and for including me in so many of the events and topics that have made graduate school a great experience.

One of the best and most lasting treasures received from my years at college is the valued friendships that I've received. I would like to thank all my friends, especially Neha Rungta, Byant Cutler, Sam Curren and Brian Whitmer.

To my parents, Douglas and Gwenda Daley and all my great family. They've been loving and so supportive and I can never thank them enough.

Thanks to my wife Cheree and my children, William, Maxson and Gwen. They have never known a father who wasn't in school. I'm glad to have made it, and it would never have been possible without Cheree's encouragement, love, support, faith and patience. I love you, Cheree.

Contents

List of Figures

Chapter 1

Introduction

Business and social interactions require that all parties involved have a solid basis for trusting one another. Knowing who you are dealing with is fundamental to building that trust. Specific facts or identity attributes are required, but they must be exchanged in a manner that preserves the integrity of the data and clearly identifies trusted sources. In the physical world we often rely on recognizing faces, hard-to-forge seals and documents, or on vouching of a trusted individual to verify identities and identity information. Unfortunately, naïve digital versions of these trust sources are intrinsically easy to forge or copy.

Claims are assertions made by a party or individual that provide the values of *attributes* for a specific *subject*. Often the attribute values in a claim are verifiable—a trusted third party (a *certifying party*) can vouch for their truth or accuracy.

One example motivating the need for attribute verification is that of a student attempting to obtain an academic discount for computer software or hardware. Any user can tell the retailer (or *service provider*) that he or she is a student— this is a *claim*, an unproven assertion of the value of that user's "student status" *attribute*. In order for the service provider to trust the value of an attribute, it needs to *verify* the attribute. To verify this attribute, the service provider might require a school the alleged student is attending to vouch for the student. In the physical world, this is usually accomplished using a hard-to-forge credential, like school letterhead or a special seal. Online, this is more difficult, since students typically lack the understanding of cryptography necessary to produce a hard-to-forge digital credential.

The financial industry is rich with examples of services that require verified attributes. For example, during a loan approval process a bank needs to verify applicant attributes including the applicant's salary, address and employer, to mitigate risk. Currently, when visiting the website of a bank, applicants can make claims by providing values for these attributes to the bank, but they have no online mechanism to allow the bank to verify these attributes. The verification is performed offline since there is no approachable and pervasive digital means of attribute verification. As traditional interactions in the real-word move online, the number of services that require verified attributes will increase.

Public key cryptography solves the technical problem of verifying claims. Certifying parties can digitally sign attribute data allowing any party to verify the attribute by verifying the digital signature. Systems which are based on public key cryptography (PKI) require significant cost in terms of infrastructure, keys, secret management and wholesale software installation. Such systems have yet to provide an adequate abstraction above cryptographic primitive functions to enable lay users to understand them and execute them securely [31] [18]. This disconnect is a failure of risk communication; users lack the contextual knowledge to make good security decisions [6]. PKI systems for attribute verification are not widely deployed or available.

Failures in effective risk communication have brought about the sentiment that humans "in the loop" are the weakest link [26] and need for their interaction should be designed out of the system [10]. However, systems enabling facilities like attribute verification require authorization or consent from users to preserve their autonomy and right to privacy. Users *must* be "in the loop" and empowered to make informed decisions about security and privacy tradeoffs.

Much recent effort has been channeled into developing identity systems and protocols in an attempt to simplify authentication across service providers (another identity function requiring user consent) while avoiding the drawbacks of pure PKI systems. Bhargav-Spantzel et al. [7] created a useful distinction between relationship-based and credential-based identity

systems. In credential-based systems, claims are encoded into hard-to-forge tokens with properties that allow them to be verified cryptographically. Relationship-based systems require verifying parties to communicate directly with a trusted party. Real-world analogs are drivers' licenses (credentials) and credit cards (relationships). Drivers' licenses are hard-to-forge tokens which can be verified by inspection, no call to the issuing DMV necessary. Credit cards require direct contact with the mutually trusted credit card company to authorize every transaction. Online relationship-based identity systems include OpenID[4], SAW[30] and OAuth[3] (OpenID and OAuth will be discussed at length later in this thesis). Most traditional identity systems are credential-based, like the X.509 protocol.

Camp et al. showed that users' internally induced model in their minds of how a system worked (a mental model) varied substantially between those with more familiarity and expertness in computer security. While a lay user is not expected to understand the system to the same depth as those who designed the system, the end user's mental model should be consistent with the underlying system to the degree that the user can use the system without inadvertently compromising security. Camp et al. demonstrated that the terms and concepts expressed in security systems are non-intuitive and alien since they overload terms and concepts of those more common to the populace with the same name. This mismatch in semantics leads to in-effective risk communication and degradation of the security of the system since users and designers have mismatching and conflicting mental models of proper system use.

SimplePermissions[12] applied *delegation*, following the typical social construct of authorizing another to stand in your place, to relationship-based authentication systems[11]. By utilizing a common and consistent abstract concept of delegation in system design SimplePermissions seeks to ensure that developers and users share a common perception or consistent mental model of the system. This consistent mental model facilitates effective communication of risk and can solve several use cases where security is typically compromised.

This thesis presents a pattern called *BackFlip* that can be applied to existing relationships such as between a student and a university or an employee and employer to provide attribute verification to third parties. *BackFlip* leverages the existing trust relationships between users, certifying parties and service providers by adding secure channels of communication and delegation policies to provide online relationship-based attribute verification. This thesis shows the correctness of this approach, prioritizes delegation to provide a consistent mental model for all users of the system and shows how this pattern could be applied using common technologies to existing systems today.

1.1 Thesis Statement

Identity systems that employ delegation can be used to provide sufficiently secure relationship-based attribute verification that is simpler and more flexible for online transactions than traditional credential-based attribute verification.

Chapter 2

Attribute Verification

Boiled down to its essence, an attribute is verified if a party you trust "says so". For example, most Department of Motor Vehicles in the United States accept a utility bill to verify a person's physical address. The DMV acknowledges the utility company as a trusted party. This makes sense since the utility company actually provides a service at a particular physical address, the DMV is leveraging the relationship between the person and the utility company to satisfy its own requirement to know the physical address of the person.

We'll employ the motivating use case of a student attempting to obtain an academic discount for computer software or hardware. The retailer (*service provider*) desires to verify potential customer's identity attribute of student status. The student (*subject*) provides a *claim*, an unproven assertion of the value of the subject's student status. To verify a claim, the service provider will require the school where the alleged student is attending to vouch that the subject is indeed a student. In this scenario, the school is a *certifying party*, a party trusted by the service provider to give the true value of a subject's attribute. The service provider may have a direct trust relationship with the school or it may use its web of relationships to construct a trusted path of relationships.

In order for a claim or attribute to be verified, the following components must be valid:

1. **Certifying Party**: A third party trusted by the service provider to give the true value of a subject's attribute. Often this is the source of the attribute about the subject.
2. **Subject**: The party or individual that is the subject of the attribute or claim.

3. **Data**: The actual value of the attribute e.g. this subject is a student.
4. **Authentication of Certifying Party**: The service provider must have proof that the assertion originated from the certifying party and not an impostor.
5. **Authentication of Subject**: The individual requesting service corresponds to the individual specified as the *subject* of a claim.
6. **Message Integrity**:The service provider must have proof that the assertion has not been modified in transit.

Conversely, if a protocol or system transfers data while exhibiting these properties, that data can be considered verified.

Identity systems are made of several components. Common to all identity systems is a means of authenticating a user. Authentication is accomplished by having the user verify some secret, like something they have, something they are, or something they know. In effect authentication is verifying a single attribute.

New identity systems like OpenID and SAW authenticate users by verifying a users claim on a globally unique identifier, a URI or email address. This makes identity verification simpler and more accessible than providing credentials based on PKI. These protocols are simple because they don't try to do everything, but are "good enough" [24] for many online use cases. In essence, OpenID and SAW allow the verification of a single attribute: ownership or control of a globally unique identifier.

OAuth is gaining widespread adoption since it enables authorization of services without requiring the disclosure of user passwords. This thesis will distill the principles of attribute verification and demonstrate how to create an environment in which OAuth can be utilized for attribute verification.

2.1 Credential-based Attribute Verification

Credential-based attribute verification is best exemplified by the X.509 protocol. X.509 is a data exchange format for certificates with a specification on which attributes should be included and how certain fields are computed and cryptographically signed. X.509 provides a simple mechanism for organizations to expose a directory. In its most primitive form, a X.509 directory entry associates a public key with an individual. X.509 certificates are generated by a Certificate Authority (CA) and contain identity information of an individual. Version three of X.509 incorporates *profiles* where additional required attributes can be enumerated.

Standard data included in certificates is the user's unique identifier or distinguished name (DN) in X.509 parlance. Since X.509 is an application of public-key cryptography each subject or user must maintain a valid public/private key pair. The user's public key is a required field in the certificate, allowing third parties to verify the bearer of the certificate is the same as specified by the Certifying Authority (CA, our certifying party) in the certificate. All of the identity attributes included in the X.509 certificate are digitally signed in aggregate with the CA's private key.

Motivating use case The certifying party (the university) where the student is enrolled creates X.509 certificates for all students. The university chooses an X.509 v3 profile to encode the attributes into the certificate.

1. **Certifying Party**: The student's X.509 certificate is digitally signed with the university's private key.
2. **Subject**: Each X.509 certificate relates to a single student, identified by the student's public key.
3. **Data**: The student's status at the university is recorded as an attribute as per the codified profile.
4. **Authentication of Certifying Party**: The retailer can know that the certificate was created by the university by verifying the signature of the certificate, that it matches

the university's public key, and that the university's public key has been signed by a trusted Certificate Authority.

5. **Authentication of Subject**: The student proves they are the subject referred to in the assertion by generating a challenge-response utilizing their private key that can be verified with the associated public key in the certificate.

6. **Message Integrity**: Attackers cannot modify the value of attribute data without invalidating the digital signature. A valid signature signifies that the certificate has not been tampered with.

Defining policy. There is no explicit policy defined by the student specifying who can access their verified information. Implicitly, the student authorizes a party to view the student's information by giving them the credential.

Selective disclosure. All attributes on the certificate are readable and verifiable. By providing the certificate, the student has disclosed all the information on the certificate. To have finer granularity of disclosure, multiple certificates would need to be manufactured by the university. The student cannot derive more granular certificates, it must be provided by the certifying party. This particular failing of X.509 has spurred work attempting to provide finer granularity of attribute disclosure using certificates [14]. Certificate-based technologies still require a large infrastructure and significant penetration of public-key cryptography.

Usability. The student must keep the X.509 certificate and the associate private key secure. The X.509 certificate must be kept safe to not divulge private information, and the private key must be kept safe so that the student cannot be impersonated. Typical computer users are incapable of correctly applying cryptographic primitives [31] that are required for day-to-day use of X.509 for attribute verification.

Discussion. While credential-based systems like X.509 are capable of providing a solution to our motivating use case, the infrastructure required by the certifying party to manage, issue, update and revoke certificates for a population is daunting. Students should be able to provide attributes to a service provider without disclosing more information than necessary.

Certificates at a smaller granularity further complicate the administrative cost of the solution and make interoperability difficult (as certifying parties and service providers would all need to support a significantly more profiles). Additionally, the administrative load on end users is unacceptable for usability reasons.

2.2 Relationship-based Attribute Sharing

The distinction between attribute sharing and attribute verification is analogous to claims and attribute verification. Attribute sharing is providing a means that values of attributes can be transferred to service providers for convenience to the end user, but not necessarily in a way that allows those attributes to be verified.

Users have existing relationships with service providers rich with attribute data that could be desirable for the user to share with other services. Examples include financial records by banks, medical records at a doctor's office, employment information by an employer, student status by a university, preference data by identity providers among others. Attribute sharing is relationship-based if the destination service provider directly contacts another service online to obtain attribute data. A popular application of relationship-based attribute sharing is for online services to provide some value-add by "mashing up" user's private data with another service or data. Some interesting examples of these mashups include online financial applications that fetch bank statements to provide reporting and budgeting tools such as Mint.com [1] or services employing facial recognition enabling bulk sorting of photos from social networking sites like PhotoTagger[2].

In OpenID, identity providers can implement an optional extension to the base authentication protocol called Attribute Exchange [16]. The relationship being capitalized is the trust relationship between the user and their chosen OpenID provider. Attribute Exchange allows a user to store arbitrary attributes at their identity provider and then

[1] http://www.mint.com
[2] http://www.face.com/

selectively disclose those values to service providers. The protocol specifies how a relying party can be granted authorization from the user to update attributes back at the identity provider. Typical use cases for this feature is that a user can be saved from repeatedly entering their address information and instead store their address (an attribute) at their identity provider. This can be very convenient for users, utilizing their existing relationship with their chosen OpenID Provider. Attribute Exchange provides no means for a service provider to verify an attribute, unless the value of the attribute is a credential. Due to this drawback, Attribute Exchange has been limited to user preferences and self-asserted attributes—attributes typically non critical in nature that do not require the assurance of verification.

With the rise of mashups and service data sharing, perhaps the most common method of attribute sharing is now directly sharing user data via OAuth-enabled API calls. OAuth is a standardized authorization protocol; an OAuth-enabled service allow users to give third-party applications ("mashups") access to user data via the service's API.

While relationship-based attribute sharing protocols enable many modern Internet use cases, they typically lack two important properties necessary for attribute verification: facilities for strong authentication of certifying parties, and message integrity guarantees. These shortcoming are addressed by *BackFlip*.

2.3 Implicit Attribute Verification

Lack of a widely-adopted attribute verification protocol has forced developers to create their own ad-hoc verification methods. For example, instead of asking directly for a verified attribute, service providers ask for another attribute highly correlated to the desired attribute that is easily verified. Verifying this secondary attribute is implicit verification, in that by verifying this attribute it implies that the actual desired attribute is also verified.

Microsoft's Greatest Steal [2] is an example of implicit verification. The Greatest Steal, an academically priced software, requires the verified attribute of our use case—a

user is a current student at an academic institution. Since this is not currently feasible Microsoft selected a highly correlated attribute with student status: a .edu email address. This attribute can be verified by sending an email that requires action of the student to the asserted email account. By successfully completing the action, this shows that the user is in control of the .edu email account.

Implicit verification is what is embraced by some of OpenID's proponents when they speak of "identity projection"[32]. The argument is that if you have an OpenID identifier from Sun [9], or the AARP, being able to verify that you control the identifier (by means of OpenID authentication) implies that you are a Sun employee or receive benefits from the AARP.

The assumptions as to how these secondary attributes are managed are not involved in the protocol, but rather they are *implicitly* defined. Specifying which attributes are verifiably attached to an identifier are performed outside the protocol in user documentation or word of mouth. Such policies can change at any time and the assumptions allowing implicit attribute verification would be invalidated.

Implicit attribute verification leverages existing relationships between user subjects, service providers and unaware trusted third parties much like SAW [30]. In the .edu email address example, Microsoft is utilizing the existing relationship between a student and university as manifest through email accounts both for student authentication and student status.

Implicit verification loses its usefulness in the scenarios that stress the difference in correlation and causation. For a concrete example let's again use our running use case:

- Having a .edu address does not necessarily mean you are a student.
- Utilizing a .edu address to implicitly verify student status assumes that universities only issue university email accounts to those who are also eligible for academically priced software.

- Revocation—Student status is typically time-bounded, where policies governing email accounts may not perfectly synchronize with academic policy.
- By merely forwarding an email, a student can enable an attacker to gain undeserved academic discounts.

Applying the principles needed for attribute verification finds this approach lacking. Service providers are not obtaining the true desired attribute, and for the correlated attribute that is being verified the process lacks strong certifying party authentication and message integrity.

While implicit attribute verification can, in the right circumstances, be a useful temporary approach it is not a suitable solution. Many useful attributes have no highly correlated simpler-to-verify counterpart. Implicit verification of attributes is not appropriate in more confidence demanding applications and its undocumented nature becomes untenable with an increase in services and consumers.

Chapter 3

BackFlip

BackFlip is a pattern that can be applied to existing relationships such as between a student and a university to provide attribute verification to third parties. To verify attributes, the *BackFlip* pattern requires the SP to exercise a delegation policy and directly access a CP web service using a secure and authenticated channel. *BackFlip* can be applied to a protocol if and only if the protocol satisfies the following conditions:

1. There exists an online relationship between a user and a CP.
2. A delegation-enabled protocol that allows users to grant access to other parties.
3. A trust relationship between the SP and the CP such that the SP accepts data vouched for by the CP.
4. A secure channel between the SP and CP providing message integrity and authentication of the CP.

Given a relationship-based protocol that satisfies these conditions, the *BackFlip* pattern provides all the necessary requirements for online attribute verification.

Consider an example shown in Figure. 3.1 that shows a user(student) who has an existing online relationship with their university(CP) where they can check their own student status through a secure and authenticated channel. A computer retailer(SP) has a relationship of trust with the university in which they accept attributes such as the student status of a user verified by the university. A student can utilize the delegation mechanism

provided by the university to grant access to the computer retailer to verify their student status attribute.

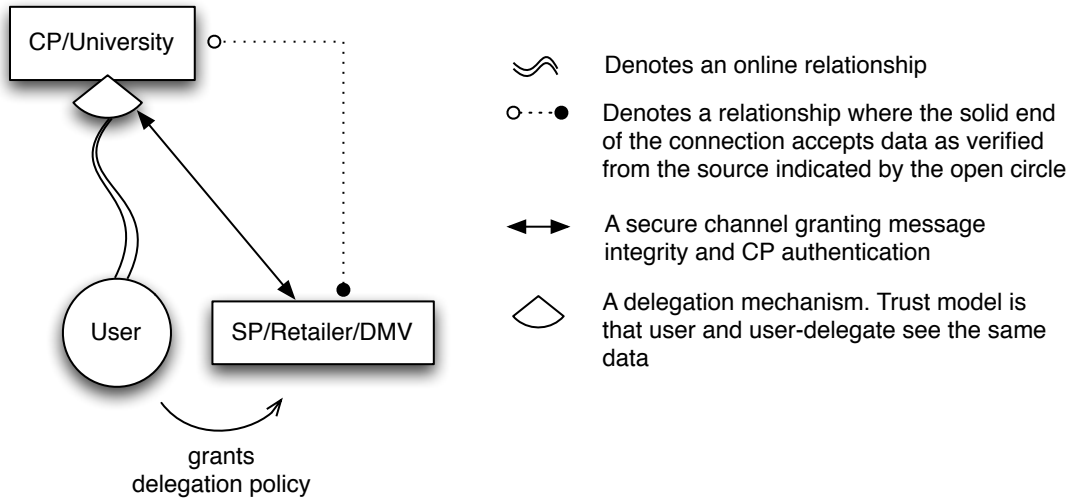

Figure 3.1: Parties and properties necessary to apply the *BackFlip* pattern.

We use an online relationship to produce relationship-based attribute verification as seen in the outer box of Figure. 3.2. Delegation-based attribute verification is a subset of relationship-based attribute verification that requires a consistent mental model to address failings in risk communication. The *BackFlip* pattern is an additional subset of delegation-based attribute verification that can progressively enhance existing applications.

3.1 Relationship-based Attribute Verification

Relationship-based attribute verification leverages the existing relationship depicted in Figure. 3.1 between a user and a certifying party to verify a user attribute to a service provider. Section 2.2 on relationship-based attribute sharing delineated how these existing relationships can be leveraged for the exchange of identity attributes and also how the properties necessary for attribute verification are not satisfied. One of the main contributions of this thesis is to show how these additional properties for attribute verification can be satisfied.

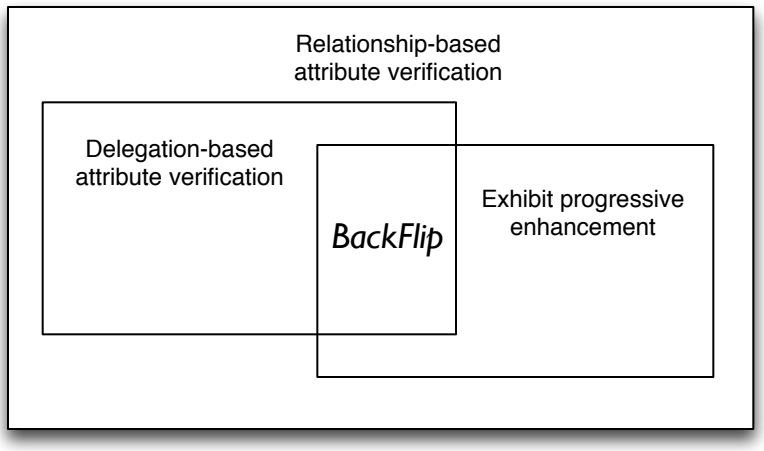

Figure 3.2: *BackFlip* as a pattern is the product of several other key patterns.

Let us assume that there is some mechanism ρ, by which the student and only the student, can define a policy granting a specific third-party access to a web service providing their student status attributes. This corresponds to the delegation mechanism provided by the certifying party to the user as shown in Figure. 3.1. When the service provider utilizes ρ in a transaction they are given access to the academic status of the student who created the policy and not any other student's status. This preserves the trust model between user and certifying party since the certifying party still only furnishes access to attribute data about the user or user-delegate requesting the resource. The function ρ provides a relationship-based system where service providers can directly contact the certifying party or originator of identity attributes as if they were the subject of an identity attribute.

More formally, as shown in Figure. 3.3, there exists a function ρ that combines arbitrary function f and authorization function g. The function ρ first calls g and if g is satisfied passes the same call to f. Let us define g as a function that permits pass-through if and only if the requesting party = (student | student-delegate). The function ρ is an example of the DecoratorPattern [13] by extending base functionality of the underlying application f by additionally allowing student-delegate.

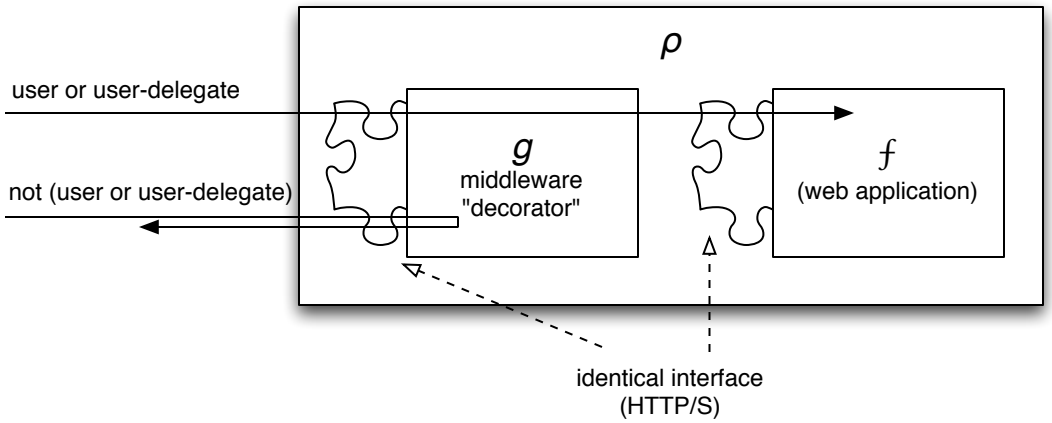

Figure 3.3: The function ρ combines an authorization function g to existing web application f

Attribute Verification To verify the attribute, the service provider requests the user to delegate access to the desired attribute data at an acceptable certifying party. The service provider directly contacts the certifying party using a secure and authenticated channel, authenticates as a user-delegate, and due to the user delegation policy is granted access to the desired identity attribute data. The service provider now has the value of the attribute, and all necessary assurances to accept the attribute as verified.

Motivating Use Case The certifying party (the university) where the student is enrolled would create a web service f, with a delegation function g available over a secure and authenticated channel.

1. **Certifying Party**: Since the certifying party is directly accessed, it is identified as the target of the request.
2. **Subject**: The subject of the attribute is implicitly declared due to our assumed user-policy-defined mechanism ρ provided by the certifying party.
3. **Data**: The student's status at the university is recorded in any format acceptable to the provider and consumer and appropriate for the purpose.

4. **Authentication of Certifying Party**: The service provider can know that they are retrieving data from the certifying party since the certifying party is authenticated by the secure and authenticated channel.

5. **Authentication of Subject**: The user proves they are the subject of the attribute data by granting access to the certifying party. If they were not the user in question they would lack the necessary permissions to delegate access to their identity information.

6. **Message Integrity**: Attackers cannot modify the value of attribute data passing between the certifying party and the service provider without being detected by the secure and authenticated channel.

We have now demonstrated that relationship-based attribute verification meets all the requirements for attribute verification similar to credential-based systems. However, there is a need to address the failings in usability. Next we show that by providing a cognitive mental model that is easy for users to conceptualize we can get a system that allows users to make informed security decisions.

3.2 Delegation-based Attribute Verification

Delegation-based attribute verification is a subset of all possible relationship-based verification protocols as seen in Figure. 3.2. Delegation-based attribute verification requires the protocol to adhere to a common abstraction or model. This common abstraction is key to addressing risk communication shortcomings in credential-based systems.

For effective risk communication, the system *metaphor* and vocabulary should elicit equivalent mental models from all participants [6]. Delegation is an ideal metaphor for risk communication in the context of attribute verification and disclosure. Application of the metaphor is constrained in this context—authorization only to access a service on one's behalf. This ensures that the metaphor doesn't "leak" or become inconsistent between all

parties involved in the transaction, from the users disclosing identity data to the software developers and protocol designers. Delegation reuses a pre-existing social construct and constrains the security protocol to interact in accordance with the metaphor so that it is not subject to the pitfalls of specialized security vernacular.

Goals of the delegation abstraction in terms of mental models are that *1*) all participants can visualize/reason about each party in the exchange as human actors; *2*) each party, like a human actor, can stand in one's place or go in one's stead; and *3*) the abstraction or metaphor closely matches the social construct of someone else representing oneself, i.e. a lawyer, trusted friend, or a co-worker filling in for you.

Target Perspective of Mental Model by Role	
User	• A service provider will be able to act as "me". • Delegation will enable the service provider to directly "speak" with a trusted source that can vouch for "me".
Service Provider	• They will ask the user to delegate access to the desired attributes at the certifying party. • They will "speak" directly to the certifying party as if they were the user.
Certifying Party	• Discloses information normally to users *and* to those authorized by the user.

Delegation provides additional context because the user already knows what information the delegate will view since it is the same information disclosed to the user. Since the delegate goes through the same door as the user and not a mysterious service entrance the user is fully aware of the data that can be revealed and make rational security decisions [17].

Since we can reason about each party in an attribute verification exchange as a human actor; the policies defined by a user are analogous to the principle of least privilege [23]. The principle of least privilege for system access has its corollary in user privacy policies: the

principle of least disclosure. Delegation allows *BackFlip* to use the principle of least privilege to effect the principle of least disclosure.

For user privacy considerations, a delegation policy embodies the user's *consent*. Delegation does not modify the trust relationship between the CP web service and the user. In the university example, the university is liable by law for releasing private data to someone not authorized by the user. In order to delegate, a user must define a delegation policy to grant another party access. This delegation policy embodies the users consent to share access to private data.

Delegation solves the issue of authenticating the user subject of a claim. The trust model is that the certifying party releases private data to the user, or to their authorized delegate. The service provider is accessing a service as if it were a specific subject, so data released by the certifying party in such a transaction is tied to that user subject.

3.3 Progressive Enhancement

We've shown that a delegation mechanism and a secure and authenticated channel can be leveraged to provide online attribute verification. We'll demonstrate here that the requirements necessary to apply the *BackFlip* pattern can be satisfied with off-the-shelf technology common to the Web today like HTTP middleware or TLS/SSL.

3.3.1 TLS/SSL

TLS/SSL satisfies the secure, authenticated channel requirement necessary to apply the *BackFlip* pattern. TLS/SSL is available in every modern browser and operating system and provides the foundation of trust for online financial transactions. SSL certificates have been successful because they allow web clients to authenticate the identity of remote web sites without requiring lay users to install, configure or directly use cryptographic functions. If both parties in a TLS/SSL handshake present valid and trusted certificates, they can

mutually authenticate. TLS/SSL can also function with just the server's certificate. This is the more common scenario on the web today.

TLS/SSL is a transport-layer protocol that can provide a private channel of communication between a client and a server set up to use an SSL Certificate. Use of a certificate provides the client with the means to authenticate the presenter of an SSL Certificate by verifying each digital signature in the chain of Certificate Authorities.

While secrecy of the communication is a nice benefit of utilizing TLS/SSL, the key characteristics of TLS/SSL that enable relationship-based attribute verification is the ability to authenticate the owner of the SSL Certificate (certifying party) and to verify the integrity of the data exchanged.

The pervasiveness of TLS/SSL is about to get better with Server Name Indication (SNI) [8] as it will allow even virtual hosts to utilize their own TLS/SSL certificates. Recent progress in upgrading the HTTP protocol could make this protocol ubiquitous. SPDY[5], an experimental protocol by Google for replacing HTTP with similar semantics but with performance of modern applications, is based on TLS/SSL and also innately satisfies the secure, authenticated channel necessary for *BackFlip*.

3.3.2 Delegation

A delegation mechanism is a key component necessary to in order to apply *BackFlip*. Delegation is a common use case and facilities are provided in a variety of protocols. Probably the most widely used means of delegation is password sharing. Dartmouth added delegation to SSL certificates [25]. Identity protocols such as ID-WSF [29] and CAS [1] provide delegation through user defined policies. Capabilities-based protocols such as ***E*** [20] provide simplified delegation mechanisms which can form the basis for authorization-based access control [19]. SimpleAuth [11] is an extension of SimplePermissions [12] that added delegation to OpenID and SSRP [15]. OAuth [3] was designed specifically to enable users to authorize or dele-

gate access to other systems and services. This work will focus on two specific delegation mechanisms, OAuth and OpenID augmented with delegation through SimplePermissions.

3.3.3 HTTP Middleware

HTTP middleware can be applied progressively to existing web applications without necessarily requiring significant modification to the underlying application. Middleware can satisfy the progressive enhancement requirement of the *BackFlip* pattern.

Even without documented APIs, many desired attributes are available through web accessible services. For example, universities provide online web services for students and staff that convey the student's academic status and progress (enrollment, matriculation, financial aid, grades etc.). While this data is not necessarily exposed via a documented API, it is addressable by its URI and is encoded in at least an HTML representation.

The architectural design of the HTTP protocol amends itself well to agglomerating various "layers" of application logic into a single unified service or application. A common use case utilizing HTTP Middleware is to extend a web application to accept TLS/SSL or HTTPS requests. The common approach to most web applications is to design the application with a web server such as Apache as the public facing endpoint that clients will access. This web server may service clients directly, or dispatch the request to either a back-end application server or an in-process program that makes up the application logic. One configures the web server to also accept connections according to the HTTPS protocol, and potentially reuse the same application logic and even the same dispatch rules of the application.

This layering of application logic allows the addition of layers nearer to the client to provide significant functionality without necessarily requiring modification of application layers further down the line. Another example of middleware is for user authentication. Some applications are architected with an explicit authentication middleware or tier, while others could be extended to use such. As introduced in section 3.2, Figure. 3.3 shows an au-

thentication middleware ρ that inspects each web request to determine if the request is being performed by an authenticated user. If the user is authenticated, the middleware proxies the request to the original application. Otherwise the authentication middleware asks the user for their authentication credentials. Delegation middleware extends the authentication logic to include user delegates and possibly implement policy defining logic.

3.4 Threat Analysis

There are two types of threats against *BackFlip*. The first threat is against the assumptions it makes concerning the online relationship between the user and the certifying party. For example, the threat to adding verification to extant data exchanges is that a service provider may accept an attribute as verified when the certifying party is merely passing the value of an attribute and not necessarily vouching for the attribute data. Secondly, vulnerabilities against an implementation or instance of *BackFlip* depend on which specific pieces of technology were chosen from those discussed in Progressive Enhancement. In the implementation section, we will choose several technologies and progressively enhance our running use case of a student attempting to gain an academic discount at a computer retailer, and analyze the threats of the specific technologies chosen.

3.5 Implementation details

To demonstrate its simplicity and flexibility; we will show how the *BackFlip* meta-protocol can be applied to two extant identity services.

3.5.1 OAuth

The specific technologies chosen to apply *BackFlip* in this example are OAuth for a delegation mechanism and TLS/SSL to provide a secure, authenticated channel to the university/CP.

This assumes there exists a web service accessible to the user at the university, and a trust relationship between the university and the retailer.

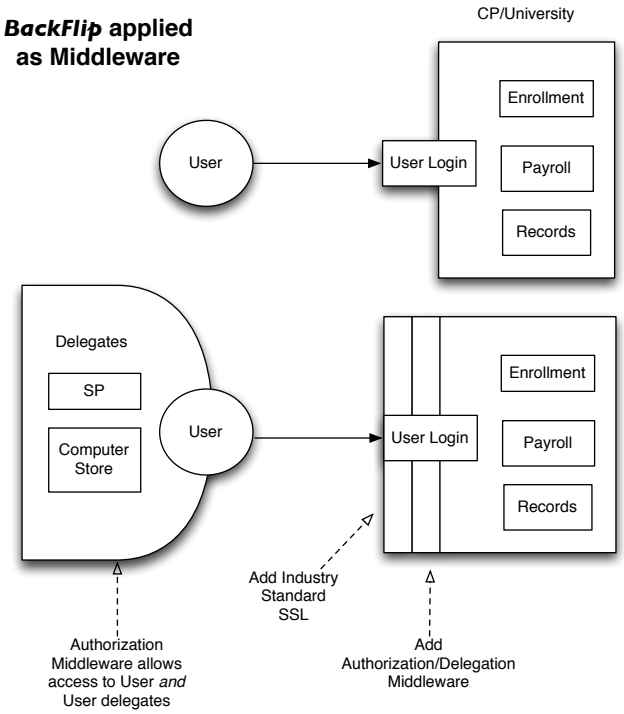

Figure 3.4: Overview applying *BackFlip* meta-protocol as middleware

Figure. 3.4 demonstrates the current state of the relationship between a user and the university. After successful authentication, the services provided by the university disclose the private user information only to the user herself including records regarding student status. In the lower section of the diagram, we show how the remaining requirements to apply *BackFlip* can be added progressively by enabling TLS/SSL and augmenting the web application with a piece of delegation middleware. The OAuth protocol can be implemented as HTTP middleware and was created for the express purpose of securely allowing users to delegate access to other service providers.

OAuth [3] is a popular protocol for web applications to gain user granted authorization to other services. OAuth enables mashups—applications that provide functionality by composing data or capability from multiple web services. Although OAuth is a new protocol,

its popularity demonstrates that delegation to third parties is a typical use case on the web today.

There are two distinct operations that OAuth performs, authorization and authentication. Authorization is the flow of messages between a provider (CP), consumer (SP) and the user that enables the user to tie a delegation policy to an authorization token associated with a service provider. Authentication is a separate exchange of messages which is performed when the consumer/SP exercises the delegated authority by exchanging an authorization token for an access token in order to access a protected resource.

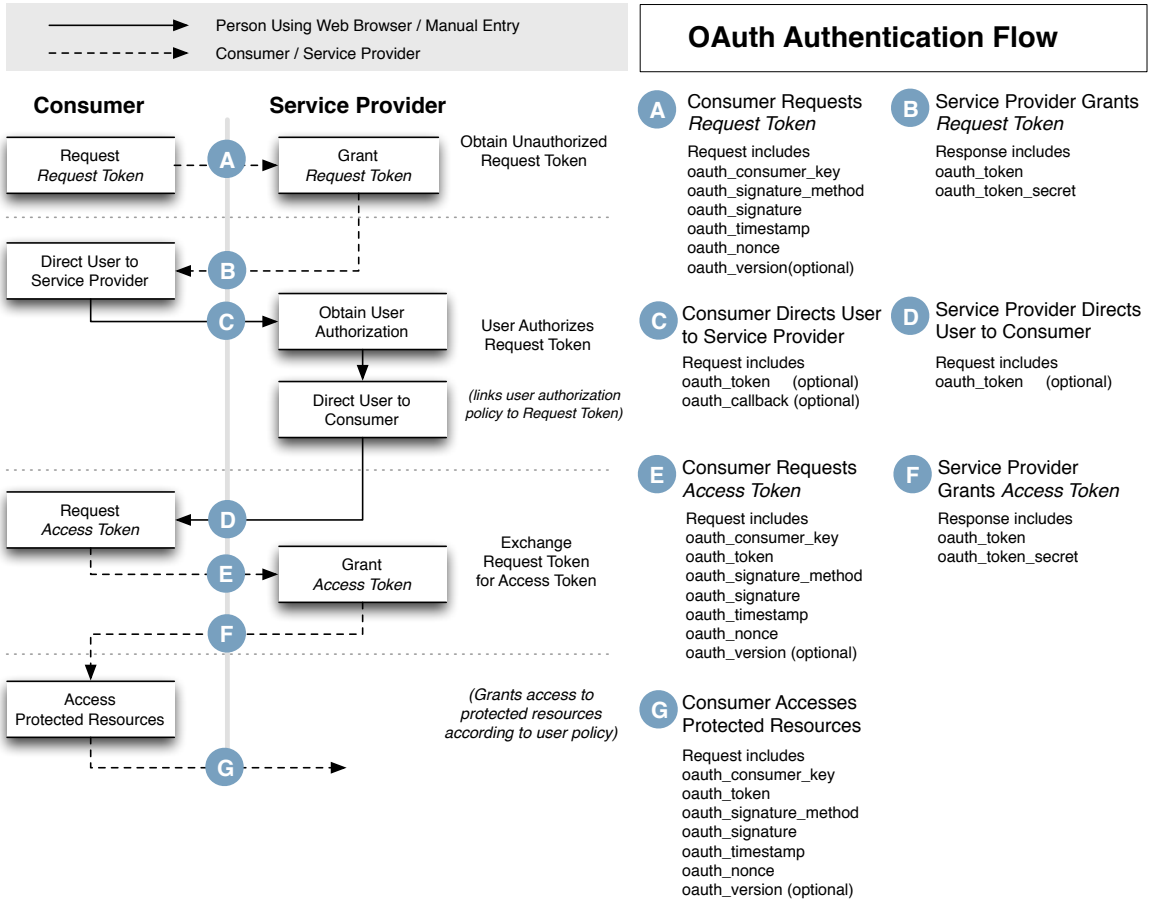

Figure 3.5: Current OAuth Protocol [3] (Adapted)

As exhibited in the steps between messages labeled C and D in Figure. 3.5, the provider/CP obtains the user's consent and authorization for the consumer/SP. This format of this request for user consent is not dictated by the OAuth protocol. In current practice

this step is largely informational: the provider typically either lists the APIs to which the consumer has requested access or asks users to generically "share their information" with the consumer. This lack of a fine-grained permission model inhibits the desired property of least disclosure/least privilege.

BackFlip improves on the usability and delegation mechanisms of OAuth for the attribute verification use case by using the permissions model of SimplePermissions as the mechanism for expressing and persisting the user's consent. Because the enhanced flow fits within the current unaltered specification of the OAuth protocol, as shown in messages labeled C and D in Figure. 3.6, the improved delegation model can be applied progressively to existing systems.

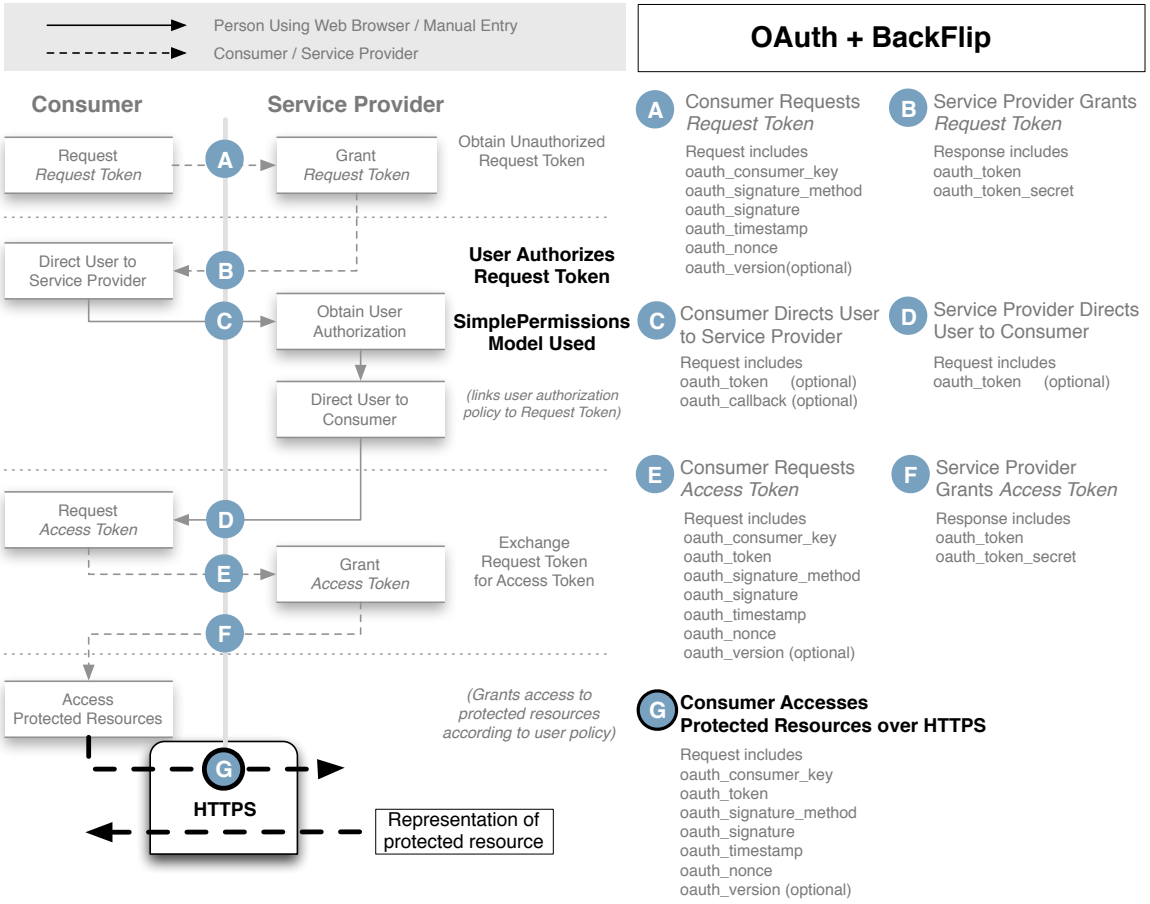

Figure 3.6: *BackFlip* augmented OAuth Protocol

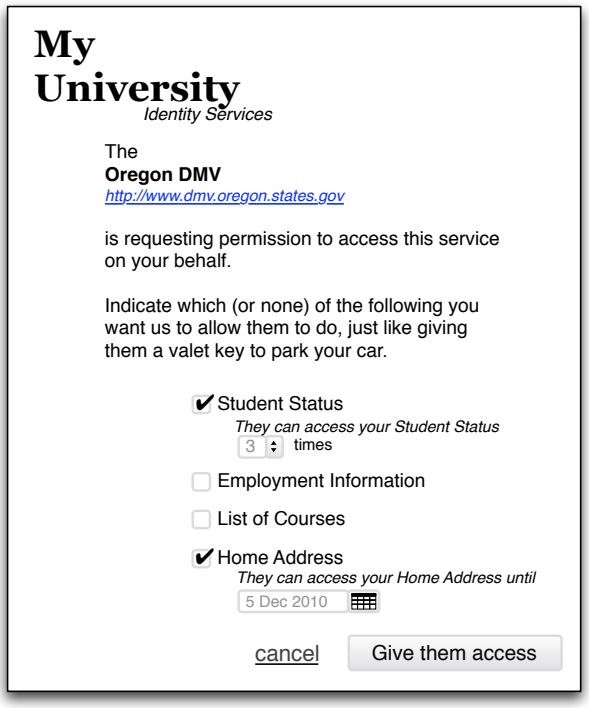

Figure 3.7: Potential interface for allowing users to define delegation policies.

The SimplePermissions permissions model can be easily extended to include time or event based predicates as shown in Figure. 3.7, like "allow access to student status... 1 time" or "allow access to my home address... for the next two weeks". Note that these predicates are not defined as a SimplePermission themselves, but are operators that modify or combine the given SimplePermissions. This is done at policy definition and can be a value-add for the identity provider allowing providers to compete with others in terms of flexibility and usability for end users. A potential example of this is manifest in Figure. 3.8 where the CP is demonstrating exactly what data will be accessible to the delegate/SP.

The subset of permissions delegated to the consumer via policy definition is stored by the CP and associated with the consumer *access token*. The access token is already opaque and unique to each user authorization policy.

Verifying an Attribute The consumer commences the unmodified authentication portion of OAuth to access the student-status protected resource. It creates a request for the specified

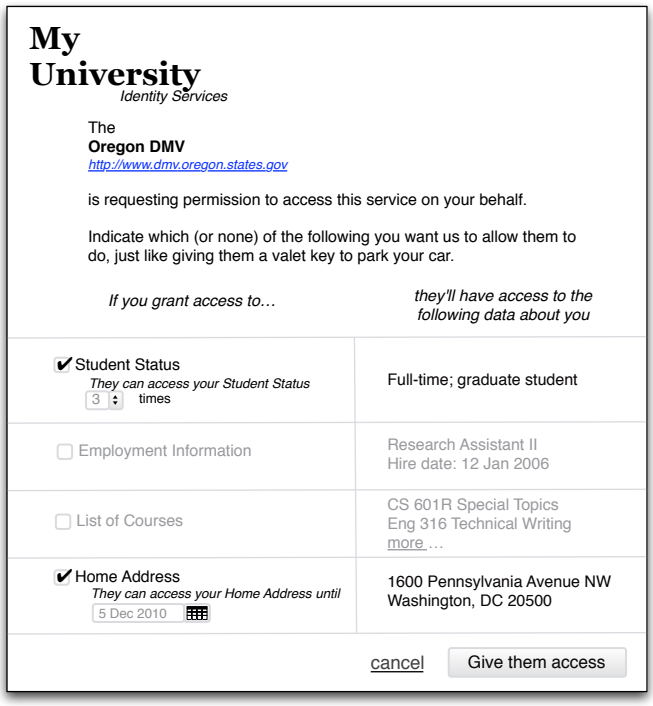

Figure 3.8: Potential interface for allowing users to define delegation policies by showing what specific data *values* will be accessible to delegates.

attribute data resource, includes all necessary data dictated by the OAuth protocol and signs the request. As indicated by step G in Figure. 3.6, this request is made over TLS/SSL. Upon reaching the CP, the associated user-defined access permissions policy is retrieved and consulted to determine if the requested access is permitted. If permitted, the CP delivers the attribute data over the secure channel to the SP. OAuth-*BackFlip* web service calls over TLS/SSL have all the properties necessary for secure attribute verification, as described in section 2.2:

1. **Certifying party:** The provider of the OAuth-enabled web service
2. **Subject:** The user who delegated the web service access
3. **Data:** The result of the API calls
4. **Authentication of the Subject:** Implied by successful OAuth access

5. **Authenticating the Certifying Party:** Provided by proper client-side use of TLS/SSL

6. **Message Integrity:** Provided by TLS/SSL

Threat Analysis

Two parts to the threat analysis; transport layer, ssl and specific to OAuth.

Transport Layer Since *BackFlip* relies on TLS/SSL to prove that the data exchange between parties is indeed from the CP, *BackFlip* is susceptible to failures in the TLS/SSL protocol. Additionally, the SP could be using the public key of the CP certificate to identify the CP. In appropriate use cases the SP could use DNS and DNSSEC to retrieve the CP identity.

TLS/SSL is the de-facto industry standard for encrypting data exchanges between two parties on the Internet but must be used appropriately to give assurances of data privacy and integrity. Recent attacks on TLS/SSL involve using certificates based on compromised hashing functions [28], or in browsers being subjected to man-in-the-middle attacks by the attacker obtaining a valid Certificate Authority certificate from one of the web browser's trusted root certificate providers [27]. Service provider SSL libraries should verify the certificates provided by the CP, that they validate, and that they are consistent with certificates previously provided by the CP. These threats against *BackFlip* are similar to any credential-based protocol in use that also depend on SSL certificates.

General threats to TLS/SSL do not necessarily apply to the controlled subset of exchanges required by *BackFlip*. For example, the SP is not required to trust the same root certificate authorities as the browsers mentioned in the related work. The underlying trust model of *BackFlip* is that the SP already has a relationship of trust with the CP, enough to accept assertions of attribute values, that the SP can obtain a SSL certificate from the CP that may or may not be signed by a root certificate authority. The two parties may provide

certificates based on some previous exchange or out-of-band in another protocol such as DNSSEC.

Permissions Model According to the OAuth specification, each CP asks permission of each user in setting up a delegation policy. The most common permissions model currently is all-or-nothing where the user agrees to give the consumer full access or no access. Adoption of the SimplePermissions permissions model and supporting user interface guidelines is within the scope of the existing specification and does not produce any additional attack vectors against the protocol.

Parameter Stealing Utilizing TLS/SSL to access protected resources does not alter the semantics of OAuth. OAuth was designed for the parameters to be included in the resource request be sent in the cleartext HTTP request. Sending these same parameters in an encrypted and integrity checking channel does not compromise needed security characteristics.

Compromises in the Delegation Mechanism Use of the authentication portion of the protocol is orthogonal to the security characteristics of the authorization portion of the system. If an attacker successfully compromised the delegation mechanism, they would be able to erroneously access protected resources. By accessing the protected resource in a manner consistent with *BackFlip*, the attacker could however, obtain the additional assurance that the transferred attributes are also verified.

Applying *BackFlip* to the OAuth protocol requires no additional delegation mechanism be provided. However, the application of *BackFlip* to a relationship-based protocol could make the enticement for breaking a given authorization protocol greater.

In the case of identity attributes being transferred in the clear by the systems described in the relationship-based attribute sharing section, the contribution of *BackFlip* is not to restrict the exchange of identity attributes, but rather to furnish existing exchanges confidence in the received data sufficient for verification or evidence.

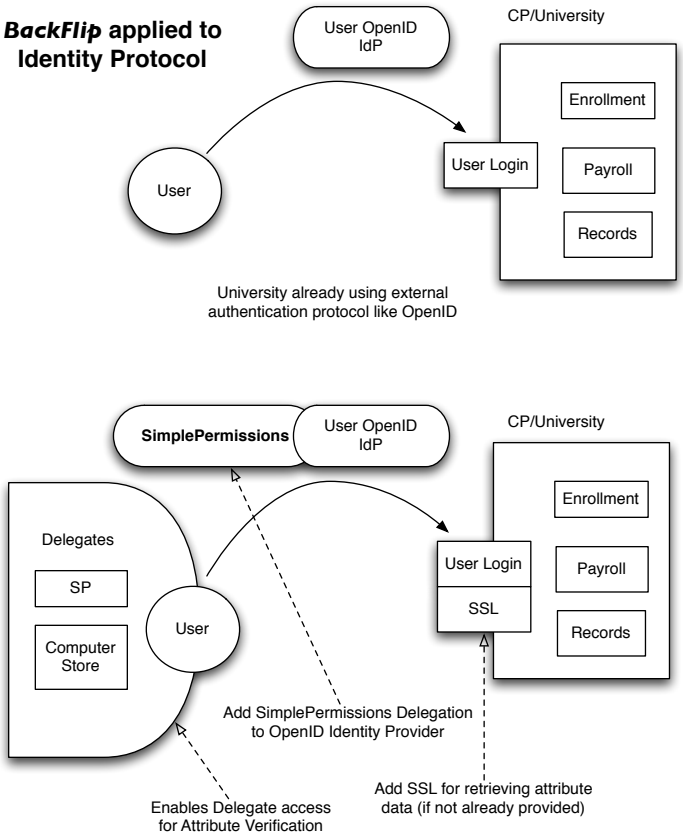

Figure 3.9: Overview applying *BackFlip* meta-protocol to the OpenID identity protocol

3.5.2 YouCanBeMe.com

YouCanBeMe is an identity proxy. An identity proxy is a identity provider that can be used as a proxy for the identity you want to use, while still using the identity as the source. A proxy acts as an identity provider to the certifying party, and as a relying party to the true identity provider. This progressive enhancement permits any OpenID users to augment their existing identities with features at the identity proxy, i.e. giving users the ability to delegate access to their data for the purpose of attribute verification.

Figure. 3.9 first depicts the existing relationship the university/CP has with the user. The user has an online relationship, it accesses a web application directly where the university

divulges data specific to the user an none other. The university is utilizing an identity protocol like OpenID to authenticate the user. *BackFlip* can be applied to provide attribute verification by adding a delegation mechanism to the identity protocol itself, as depicted in the lower portion of Figure. 3.9.

OpenID is a relationship-based authentication protocol where users log in to sites that adhere to the OpenID specification by providing a discoverable identifier like a URI. An OpenID relying party relies on the OpenID identity provider to authenticate and vouch for the user without directly accepting any authentication credentials. OpenID does not have a facility for user to user delegation natively. SimplePermissions not only provides a delegation mechanism for OpenID, but establishes a simple permissions model in the format of boolean predicates which lead to a simple policy interface and selective delegation. OpenID and SimplePermissions are covered in more depth in Appendix A and Appendix B.

YouCanBeMe is a re-implemention of the SimplePermissions work in OpenID. The implementation extends previous work with OpenID 2.0's directed identity. YouCanBeMe also crosses over into other identity protocols by allowing users to claim and link multiple identities and protocols with their account. This means that they can create the account using their email as an identity (using SAW), and additionally claiming their Twitter identifier (which is done via OAuth) or Facebook (OAuth2). They can then log in to YouCanBeMe using any one of these identities or people can delegate access to services by way any one of these identities. YouCanBeMe is a relying party in the following protocols, OpenID, OAuth, OAuth2, and SAW.

Scenario 1: A user uses YouCanBeMe to login to a site.

1. The user provides the OpenID identifier, youcanbeme.com to Cp.com where Cp.com represents any OpenID relying party that provides identity data about the user(a CP), like Jyte, or StackOverflow.

2. Cp.com performs discovery on youcanbeme.com, retrieves an XRDS document advertising support for OpenID 2.0 directed identity and SimplePermissions.

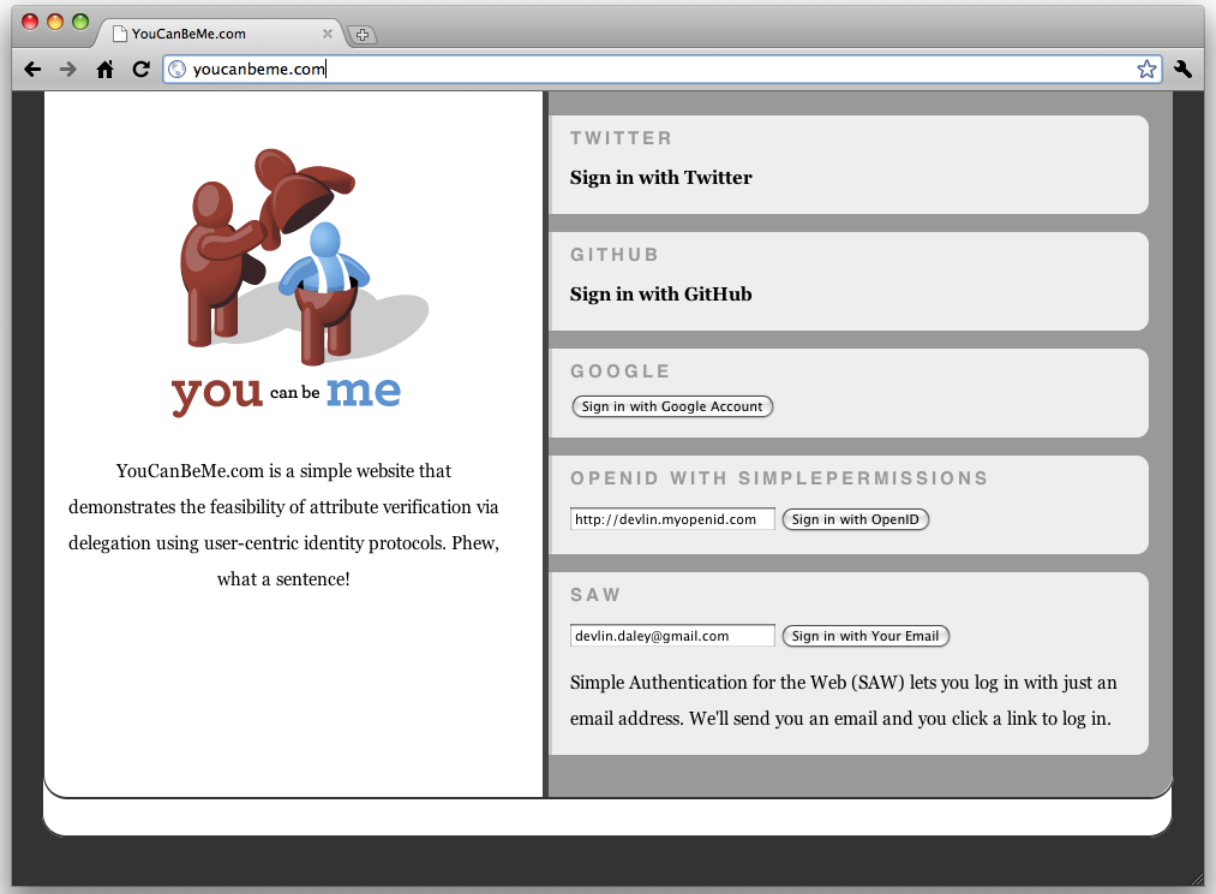

Figure 3.10: Login screen for YouCanBeMe, an identity proxy that furnishes selective delegation to OpenID that can be used for attribute verification.

3. Cp.com optionally negotiates a shared secret with youcanbeme.com for later message authentication.

4. Cp.com generates a checkid_request, attaching a permissions model of canRead and canEdit, and embeds it in a browser redirect to youcanbeme.com.

5. YouCanBeMe.com receives the request, sees that the user is unauthenticated, stores the request and the associated permissions model and directs the user to log in.

6. The user can log in with any identifier of their choosing. See Figure. 3.10. If this is their first time at YouCanBeMe, using any identity protocol will automatically create them an account. After log-in their are directed to the decision page.

7. At the decision page, YouCanBeMe verifies that the user is wanting to assert their identity information. By default all permissions are granted in the permissions model.
8. The user is redirected back to Cp.com with an id_res message and they are successfully logged in.

Scenario 2: How a user delegates access to a site.

1. A user logs in to YouCanBeMe.com using any of the identities they've associated with their account.
2. On the main page, they are presented with a list of certifying party sites they have accessed along with recent activity for their delegates.
3. They select one of the certifying party sites previously visited.
4. The user is shown all existing delegation policies to users and entities for this given site (along with any activity). To delegate, the user supplies an email address and, if Cp.com had supplied a permissions model, allows for selectively delegating permissions based off of the descriptions provided specifically for this purpose. If no permissions model was advertised, the user can still create a policy.
5. The delegation policy is created, the user is shown the email that will be sent to the delegate.

Scenario 3: The delegate exercises delegation policy to log in at a site.

1. The delegate provides the OpenID identifier, youcanbeme.com to Cp.com
2. Cp.com performs discovery on youcanbeme.com, retrieves an XRDS document advertising support for OpenID 2.0 directed identity and SimplePermissions.
3. Cp.com optionally negotiates a shared secret with youcanbeme.com for later message authentication.
4. Cp.com generates a checkid_request, attaching a permissions model of canRead and canEdit, and embeds it in a browser redirect to youcanbeme.com.

5. YouCanBeMe.com receives the request, sees that the delegate is unauthenticated, stores the request and the associated permissions model and directs the delegate to log in.

6. The delegate can log in with any identifier or identity service of their choosing. However, whichever identity provider is chosen, it must assert the same email address provided by the user when creating the delegation policy. For simplicity, the delegate may initially log in using Simple Authentication for the Web (SAW) which uses the email communication channel to verify their identity. After log-in their are directed to the decision page.

7. At the decision page, YouCanBeMe displays a list of the possible identities this delegate can assume at Cp.com. By default, YouCanBeMe creates a personal policy for the delegate so that they can log in to Cp.com as themselves, for the user use case. Additionally, they are given a list of delegation policies displaying the authorizing user and what rights were granted according to Cp.com's permissions model. The delegate selects to invoke the policy created by the original user. YouCanBeMe logs that the policy was exercised.

8. The delegate is redirected back to Cp.com with an id_res message containing all permissions asserted. They are successfully logged in as the original authorizing user.

9. If attribute data is accessed via TLS/SSL, all properties delineated by *BackFlip* for attribute verification are satisfied.

YouCanBeMe was created as a simple Ruby application using the Sinatra framework. They key pieces included creating:

1. An OpenID 2.0 Identity Provider.

2. Middleware (as a Rack component) creating a complete OpenID Relying Party, augmented with SimplePermissions.

3. Relying party middleware for SAW.

4. A persisted data model of users, multiple authentication channels, delegation policies with audit logs of activity.

YouCanBeMe implements all relying party code, including the SimplePermission delegation functionality, as middleware. Middleware shows that this functionality can be re-used in other web applications with minimal modifications. The benefits of this approach include audit trails of when delegates access authorized sites and identity information and to easily provide a revocation mechanism since delegates must pass through the user's identity provider.

One major contribution of this work is the recognition that OpenID-SimplePermissions web service calls over TLS/SSL have all the properties necessary for secure attribute verification, as described in section 2.2:

1. **Certifying party:** The provider of the OpenID-SimplePermissions-enabled web site or service.
2. **Subject:** The user who granted site access via delegation policy.
3. **Data:** The result of the API calls.
4. **Authentication of the Subject:** Implied by successful OpenID access.
5. **Authenticating the Certifying Party:** Provided by proper client-side use of TLS/SSL.
6. **Message Integrity:** Provided by TLS/SSL.

Threat Analysis

In the OpenID application of *BackFlip* as exhibited by the identity proxy YouCanBeMe, we use the same transport layer implementation as OAuth. As described in the OAuth threat analysis of section 3.5.1, the same choice of using TLS/SSL equally applies in this contenxt.

The threat analysis of an attack on the exchange of permissions models is covered by Bryant et al. [12]. In general, if an attacker is able to compromise the delegation feature of an

identity provider they could gain unauthorized access to sensitive information. However, by outsourcing authentication via an identity protocol, any compromise to the identity provider would allow *unrestricted* impersonation of any user of that identity provider. External identity providers have the ability themselves to trivially impersonate any user, so asking them to provide a delegation mechanism doesn't open any additional security vulnerabilities that are not already extant with the current trust model between a certifying party and the identity provider.

Chapter 4

Conclusion

Delivering user attribute data in a manner where the receiving party can *verify* the attribute as being vouched by a trusted party is possible using credential-based systems and traditional PKI. Unfortunately, these approaches have not found widespread adoption due to lack of end-user usability, ineffective risk communication and the necessity of wholesale modification of existing applications.

While pure PKI systems have not gained widespread adoption parts of them have; most notably TLS/SSL.

BackFlip employs a principled approach to attribute verification—re-analyzing the principles and properties necessary for attribute verification. *BackFlip* takes advantage of the fact that features baked into PKI systems were based on requirements that are no longer required. For example, PKI can work in an offline mode, meaning if it had access to public keys, it does not require network access to verify encoded credentials. This was due in part to unreliable and non-pervasive network access. This issue has been resolved with the success of the Internet and is best exemplified by the rise of relationship-based protocols where identity data is verified by directly interrogating identity providers.

BackFlip shows how relationship-based attribute verification is possible by utilizing TLS/SSL. *BackFlip* takes the next formative step by addressing failings in usability by proposing delegation-based attribute verification, and citing prior work where delegation can be added to relationship-based systems.

Delegation-based attribute verification harnesses a common social construct of delegation and applies it to authorization policy definition. This constrained social construct induces equivalent mental models between developers of application security and end-users which is imperative for effective risk communication.

As a principled approach, *BackFlip* is a meta-protocol or design pattern which was applied to several identity protocols, OAuth and OpenID. These concrete implementations show that the pattern exhibits progressive enhancement, applying both to new applications as well as legacy systems.

This work

1. Shows that relationship-based attribute verification is possible; i.e. correctness.
2. Shows that relationship-based attribute verification can be accomplished incrementally on the Web.
3. Introduces delegation-based attribute verification.
4. Demonstrates granular delegation-based attribute verification.
5. Proposes utilizing the SimplePermissions permissions model for effective risk communication in contexts of attribute verification and identity attribute disclosure.
6. Incorporates OpenID 2.0 Directed Identity into the SimplePermissions OpenID protocol extension.
7. Creates an OpenID delegation proxy, YouCanBeMe.
8. Provides concrete user interface and user experience implementations defining granular based authorization policy, applicable both to OpenID and ongoing discussions in OAuth communities.

BackFlip demonstrates that identity systems which employ delegation can be used to provide sufficiently secure and delegation-based attribute verification with several advantages over credential-based systems.

4.1 Future Work

While *BackFlip* makes solid arguments for delegation-based attribute verification, future work would include a user study to measure not only user understanding of delegation model but potentially user focused attacks against it.

Future work includes investigating the feasibility of encoding delegated authorization as a *capability* compatible with web browsers and existing services. Capabilities could potentially facilitate multi-hop verification among other benefits. Additional insights could be gained from applying *BackFlip* to additional identity protocols like SAML, CAS and Kerberos.

Appendices

Appendix A

OpenID

OpenID is a distributed and federated identity protocol that has been gaining widespread support on the web recently. OpenID furnishes users with a globally unique identifier, called an OpenID, that can be used at any consenting service or relying party. OpenID allows users to use this global identifier instead of creating separate accounts at each service. It is federated in the sense that any service or Identity Provider that complies with the OpenID protocol can both consume and provide OpenID identities.

OpenID is a relationship-based protocol. The third party trusted by both the user and relying party to mediate authentication is called the OpenID Identity Provider or IdP.

The core OpenID protocol has two main focuses, discovery and authentication. Discovery is the process by which a relying party can utilize existing web technologies to find the user's desired Identity Provider. Authentication is the process by which each relying party can receive an assurance that the user correctly authenticated with the IdP without the user disclosing their secret credentials, passwords etc. directly to the relying party.

OpenID is a single sign on solution in the sense that the user can utilize the same identifier and related credential across multiple sites. OpenID has a under-utilized mode that attempts to furnish true single sign on (where if a user is logged in to site A, when they visit site B they are already signed in) called *checkid_immediate*.

At a relying party, a user provides their OpenID identifier. Their OpenID is a string representing either an i-name [22] or more commonly a URI. The relying party performs discovery on this identifier by querying the resource located at the URI. The relying party

uses HTTP content negotiation to retrieve the resource in machine readable format. The OpenID discovery protocol suggests encoding the identifier meta-data in an XRDS document; a dialect of XML. The discovered meta-data specifies the user selected OpenID Identity Provider.

The relying party may optionally contact the IdP using HTTP and perform a Diffie-Hellman algorithm to generate a shared secret with the IdP that can be used to sign subsequent messages for message integrity.

OpenID makes the assumption that the user's client is a "dumb browser", meaning that the browser is unaware of the OpenID protocol but can correctly follow HTTP redirects and requests. OpenID leverages HTTP redirects, javascript form submissions and URL request parameters to transport messages between the relying party and the IdP by way of the user's web browser.

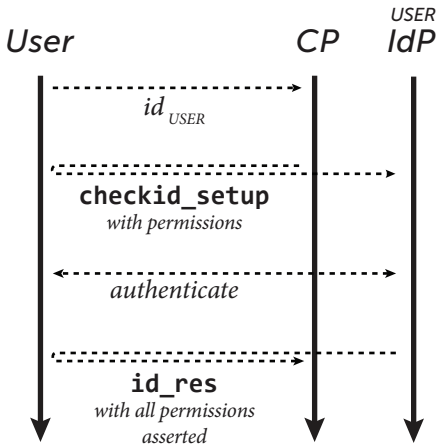

Figure A.1: OpenID protocol flow.

If the OpenID protocol advances to completion the end result of an authentication is an OpenID Response message. The message is optionally signed with the shared secret between the specific relying party and IdP or it can be submitted directly to the IdP by the RP for completely stateless operation. The OpenID specification dictates which OpenID Response messages are positive assertions and negative assertions. Essentially, the end result of an OpenID authentication is a message where the OpenID Identity Provider says "yes"

or “no” concerning whether the user successfully authenticated and granted authentication verification to the relying party.

Appendix B

SimplePermissions

SimplePermissions is a pattern for giving users the ability to delegate their privileges by granting permissions to other users. It uses a relationship-based authentication protocol as a transport mechanism for authorization information. SimplePermissions harnesses authorized impersonation to enable secure and user consented access to services.

The pattern has three main phases, *permission model discovery*, *delegation* and *delegate authorization*. While the pattern has also been applied to several relationship-based authentication protocols[11], this description is based on the OpenID authentication protocol.

BackFlip uses SimplePermissions and its associated permissions model to allow users to express access to identity data as granting permissions.

The sequence of events in normal usage is as follows:

Permissions Model Discovery

Permissions model discovery is performed when the user authenticates at a Service Provider, as shown in A.1. A full boolean based permissions model is exchanged in the locations indicated by "permissions" in the figure, instead of the implicit "all permissions" and "all functionality" assumption in the standard OpenID protocol.

1. A user begins to authenticate to a service provider using a relationship-based authentication protocol (e.g. by submitting his OpenID to a web application).

2. In the course of authentication, the service provider contacts the user's identity provider. SimplePermissions-enabled SPs include in this message a list of the privileges available to the user.

3. The list of privileges for that user at that service provider is saved by the user's identity provider.

4. The user authenticates his identity using whatever means the authentication protocol provides (e.g. with a password, Information Card, or TLS/SSL certificate).

5. In replying to the SP, the user's IdP includes the list of privileges the user wishes to exercise for the duration of the session. By default, when the user himself is authenticating his IdP asserts that he will exercise all his available privileges.

6. After the user is authenticated, he has access to all the appropriate application functionality.

Delegation

1. Later, the user decides to delegate his privileges. He visits his identity provider, which allows him to enable or disable each permission in the saved permissions model for a specific delegate. These user choices are stored as a *delegation policy* at the identity provider.

Delegate Authorization

These descriptions explain the flow of the protocol as shown in Figure. B.1.

1. The delegate user begins to authenticate to the service provider, in such a way that the authentication request is directed towards the delegating user's IdP. The delegate provides the delegating user's `openid_url`, instead of her own.

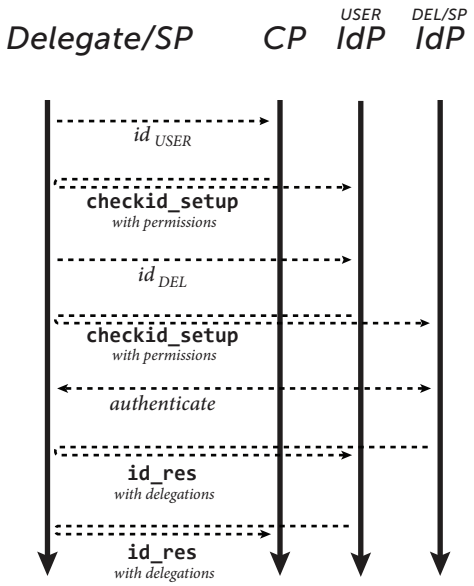

Figure B.1: Authentication of Delegate and interaction with the User's OpenID identity provider(IdP).

2. The delegate then authenticates to the user's IdP as though it were a service provider, using a relationship-based authentication protocol. The delegate provides her own `openid_url` to the IdP instead of a login credential.

3. An authorization request is sent to the delegate's IdP. The delegate authenticates her identity to her IdP using the appropriate credentials.

4. The delegate's IdP responds to the delegating user's IdP, asserting that the delegate has authenticated.

5. The delegating user's IdP consults the user's delegation policy. The delegate is returned to the SP with the list of permissions granted by the policy for the current session.

6. The delegate has authenticated, and has access to the appropriate subset of the functionality available to the delegating user.

```
...
openid.ns.simplepermissions = http://openid.net/specs/simplepermissions
openid.simplepermissions.canReadEmail="The user may read email sent to this account"
openid.simplepermissions.canSendEmail="The user may send email from this account"
openid.simplepermissions.isAdmin="The user has full administrator rights"
openid.simplepermissions.changeAccountPassword="The user logging in can change the account password"
...
```

Figure B.2: Example SimplePermissions service provider permissions model attached to a `checkid_setup` request

Permissions Model Discovery

SimplePermissions service providers advertise a permissions list to each of their authenticating users' IdPs. An example permission model, advertised in our OpenID implementation by attaching namespaced attributes to an authentication request, is shown in Figure. B.2. Each permission in the list is a name/value pair. The name of the permission is the key that will be used in a delegation response. Permission names are chosen to be associated with boolean values, and may describe a specific service capability, membership in a role, or a user attribute required for policy enforcement within the SP. The value associated with that permission name is a text description of the permission. This text description may be customized indepedently of the permission name; for example, an SP may return localised description strings based on the `Accept-Language` HTTP header of the user's authentication request.

Permissions models are designed for human consumption and not machine processing. The goal of the permission descriptions is to allow users to make informed delegation decisions. The simple boolean-values-only model encourages helpful names and descriptions that expose the internal permissions model to user understanding; more complex models that allow scalar or complex values can easily become opaque to users (e.g. a permission like "privilegeLevel=5" requires knowledge of the scalar-to-permission mapping internal to the system). The simple permissions model also allows each IdP to develop a standard user interface for users to grant permissions to delegates.

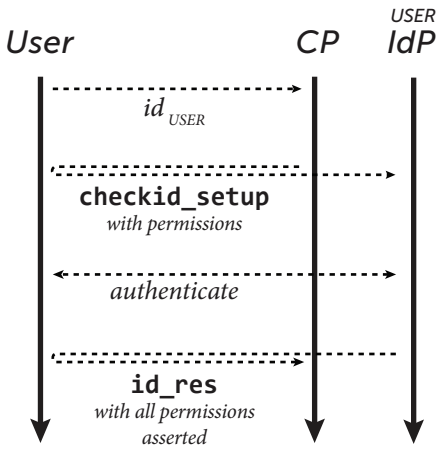

Figure B.3: SimplePermissions OpenID permissions model discovery (`associate` messages omitted for clarity)

SimplePermissions permissions are exchanged during authentication (see Figure. B.3) because the act of authenticating signals the user's intention to exercise the privileges granted by the SP. While it is possible to expose a site's permission model in other ways (via a conventional web services API or in HTTP cookies as in [25]) these do not in general allow for per-user customization.

Delegation

A user delegates his SimplePermissions permissions by creating a delegation policy instructing his IdP to assert a subset of his privileges during authentication of delegates. The mechanism for specifying and persisting policy is beyond the scope of the SimplePermissions pattern. It is envisioned that this would *typically* consist of per-delegate-and-service-provider permissions lists persisted into a relational database via a web application. However, it is possible that a SimplePermissions-enabled IdP could generate policies automatically based on the relationship graph of a social network, the organizational hierarchy expressed in an LDAP directory, or even a dynamic reputation system. The policies might be stored in an ad-hoc way, or in a standardized format such as XACML [21].

In addition to the potential user interfaces described in the OAuth section, Figure. B.4 demonstrates that the use of the SimplePermissions permissions model allows a significant range of combinations and manipulation of a service provided model by the user's OpenID identity provider..

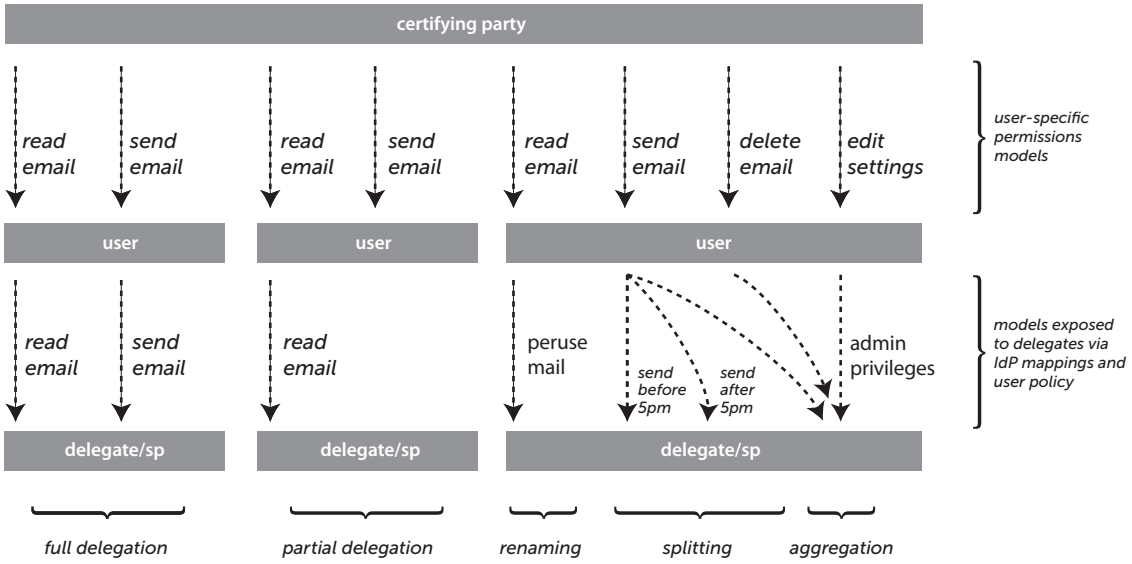

Figure B.4: Authorization modifications of SimplePermissions models.

BackFlip utilizes SimplePermission's focus on communicating first to the user to be consistent with the requirements of effective risk communication. Authorization in terms of allowing another person or product to access a service on one's behalf induces the mental model of delegation and becomes analogous with the social construct of asking another to stand in one's place. *BackFlip* applies these characteristics to both OpenID and OAuth. With an effective delegation mechanism in place, *BackFlip* satisfies the remaining properties necessary for attribute verification.

Contributions to SimplePermissions:

1. More formally defines the permissions model in terms of the rich research in mental models and risk communication.
2. Applies the SimplePermissions model to OAuth.

3. Extends the OpenID implementation to cross identity protocols. Users can delegate to OAuth and SAW identities in addition to OpenID identifiers.

4. Adds OpenID 2.0 Directed Identity.

References

[1] Central Authentication Service. http://www.ja-sig.org/products/cas/.

[2] Microsoft – The ultimate steal promotion. http://www.microsoft.com/education/ultimatesteal.mspx.

[3] OAuth core 1.0 specification. http://oauth.net/core/1.0/.

[4] OpenID 2.0 specification. http://openid.net/specs/openid-authentication-2_0.txt.

[5] SPDY: An experimental protocol for a faster Web. http://dev.chromium.org/spdy, 2010.

[6] F. Asgapour, D. Liu, and L. J. Camp. Risk communication in computer security using mental models. In *Workshop on the Economics of Information Security 2007*, June 2007.

[7] A. Bhargav-Spantzel, J. Camenisch, T. Gross, and D. Sommer. User centricity: A taxonomy and open issues. *Journal of Computer Security*, 15(5):493–527, 2007.

[8] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport layer security (TLS) extensions. http://www.ietf.org/rfc/rfc4366.txt.

[9] T. Bray. OpenID at work. http://www.tbray.org/ongoing/When/200x/2007/05/07/OpenID-at-Sun.

[10] L. F. Cranor. A framework for reasoning about the human in the loop. In *UPSEC'08: Proceedings of the 1st Conference on Usability, Psychology, and Security*, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association.

[11] B. Cutler. Simple, secure, selective delegation in online identity systems. Master's thesis, Brigham Young University, August 2008.

[12] B. Cutler, D. Daley, P. Windley, and K. Seamons. SimplePermissions: an OpenID extension for delegation and permissions model discovery. Technical Report 200801, Brigham Young University Enterprise Computing Lab, 2008.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Professional, 1995.

[14] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS '06: Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 89–98, New York, NY, USA, 2006. ACM.

[15] A. Harding, T. van der Horst, and K. Seamons. Wireless authentication using remote passwords. In *1st ACM Conference on Wireless Network Security (WiSec)*, Alexandria, VA, March 2008.

[16] D. Hardt, J. Bufu, and J. Hoyt. OpenID attribute exchange 1.0. http://openid.net/specs/openid-attribute-exchange-1_0.txt.

[17] C. Herley. So long, and no thanks for the externalities: the rational rejection of security advice by users. In *Proceedings of the 2009 workshop on New Security Paradigms Workshop*, pages 133–144. ACM, 2009.

[18] M. Johnson, C. Atreya, A. Aviv, M. Raykova, S. Bellovin, and G. Kaiser. RUST: A retargetable usability testbed for website authentication technologies. *Usenix UPSEC '80: Usability, Psychology, and Security*, 2008.

[19] A. H. Karp. Authorization-based access control for the services oriented architecture. *Creating, Connecting and Collaborating through Computing, 2006. C5 '06. The Fourth International Conference on*, pages 160–167, Jan. 2006.

[20] M. Miller. The E language. http://erights.org/.

[21] T. Moses et al. eXtensible access control markup language (XACML) version 2.0. *OASIS Standard*, 200502, 2005.

[22] D. Reed, D. McAlpin, P. Davis, N. Sakimura, M. Lindelsee, and G. Wachob. Extensible resource identifier (XRI) syntax and resolution specification. http://www.oasis-open.org/committees/xri, 2003.

[23] J. Saltzer and M. Schroeder. The protection of information in computer systems. In *Proceedings of the IEEE*, volume 63, pages 1278–1308, Sep 1975.

[24] R. Sandhu. Good-enough security. *Internet Computing, IEEE*, 7(1):66–68, 2003.

[25] N. Santos and S. W. Smith. Limited delegation for client-side SSL. In *6th Annual PKI Research and Development Workshop*, Apr 2007.

[26] B. Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley and Sons, 2000.

[27] C. Soghoian and S. Stamm. Certified lies: Detecting and defeating government interception attacks against SSL. http://files.cloudprivacy.net/ssl-mitm.pdf, April 2010.

[28] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. Osvik, and B. de Weger. MD5 considered harmful today. In *Announced at the 25th Chaos Communication Congress*, 2008.

[29] J. Tourzan and Y. Koga. Liberty ID-WSF web services framework overview. http://www.projectliberty.org/specs/.

[30] T. W. van der Horst and K. E. Seamons. Simple authentication for the Web. In *3rd International Conference on Security and Privacy in Communication Networks*, Nice, France, September 2007.

[31] A. Whitten and J. Tygar. Why Johnny can't encrypt: A usability case study of PGP 5.0. *Proceedings of the 8th USENIX Security Symposium, August*, 1999.

[32] S. Willison. Yahoo!, Flickr, OpenID and identity projection. http://simonwillison.net/2008/Jan/7/projection/.