DOI:10.1145/2366316.2366333

Decryption keys allow users to learn a specific function of the encrypted data and nothing else.

BY DAN BONEH, AMIT SAHAI, AND BRENT WATERS

Functional Encryption: A New Vision for Public-Key Cryptography

ENCRYPTION IS A method for users to securely share data over an insecure network or storage server. Before the advent of public-key cryptography, a widely held view was that for two users to communicate data confidentially they would have to first establish a mutually held secret key *k*. While acceptable, perhaps, for some small or tight-knit organizations, such a solution is clearly infeasible for larger networks (such as today's Internet). More than 30 years ago, Diffie and Hellman[11,12] introduced the concept of public-key cryptography, where two parties securely communicate with each other without having a prior mutual secret, radically challenging the conventional wisdom of the time.

Today, public-key encryption is invaluable, ubiquitous in securing Web communication (such as HTTPS and SSH), voice traffic, and storage systems. However, within the technical community, there is an ingrained view that:

- Access to the encrypted data is "all or nothing"; one either decrypts the entire plaintext or learns nothing about the plaintext (other than a bound on its length); and
- Encryption is a method to encode data so a single secret key can decrypt that data.

However, for many applications, this notion of public-key encryption is insufficient; for example, the encryptor may want to encrypt data so anyone satisfying a certain policy can then decrypt it. Consider encrypting a message to a company so the only users who can decrypt it are employees in, say, the accounting or sales departments whose office is in the company's main building. Realizing this application using existing public-key encryption raises several questions:

- How do we discover the public keys of all individuals who satisfy this policy?;
- What if someone joins the system or receives certain credentials well after the data is encrypted and stored?;
- What if we want to give someone a partial view of the plaintext depending on their credentials?; and
- Should a given user even be al-

» key insights

- Unlike traditional encryption, where decryption is all or nothing, in a functional encryption system decryption keys may reveal only partial information about the plaintext; for example, decrypting an encrypted image with a cropping key will reveal a cropped version of the image and nothing else.
- Many advances in public-key encryption over the past decade can be viewed as special cases of functional encryption.
- Current functional encryption systems are quite expressive, yet much work remains in expanding the set of functionalities supported by existing constructions.

chrome://global/skin/" type="text/
chrome://zotero/skin/zotero
chrome://zotero-platform/content/
contextDTD SYSTEM
andaloneDTD
andDTD SYSTEM "chrome://brand
="http://www.mozilla.
copyCmd
accesskey="&deleteCmd.accesskey;"
command="cmd_delete"/>
id="content-sep-paste"/>
id="content-selectall"
label="&selectAllCmd.label;"
accesskey="&selectAllCmd.accesskey;"
command="cmd_selectAll"/>
="mainCommandSet"/>
id="browser">
flex="1">

lowed to learn the identities of all individuals who have certain credentials?

Functional encryption. It is time to adopt a new broad vision of encryption that takes into account such concerns. To this end, we advocate the concept of "functional encryption" where a decryption key enables a user to learn a specific function of the encrypted data and nothing else. Briefly, in a functional-encryption system, a trusted authority holds a master secret key known only to the authority. When the authority is given the description of some function *f* as input, it uses its master secret key to generate a derived secret key sk[*f*] associated with *f*. Now anyone holding sk[*f*] can compute *f*(*x*) from an encryption of any *x*. In symbols, if *E*(pk; *x*) is an encryption of *x*, then decryption accomplishes

given *E*(pk; *x*) and sk[*f*],
decryption outputs *f*(*x*).

Note it is *f*(*x*) that is made available to the secret key holder, even though *x* was the value that was encrypted. A functional-encryption system can support a variety of functions this way. Intuitively, the security of the functional-encryption system should guarantee the secret-key holder can learn nothing else about *x* beyond *f*(*x*). We thus envision functional encryption as analogous to secure computation[18,33] but with the critical difference that functional encryption is completely noninteractive once a recipient obtains the recipient's own secret key.

Consider what is possible if functional encryption would be realized for a broad set of functions:

Spam filtering on encrypted mail. A user wishes to leverage a partially trusted proxy to filter out all encrypted email messages identified as spam according to the user's criteria. The user wants to achieve the seemingly conflicting goals of hiding the message's contents from the proxy while allowing the proxy to determine if the message is spam according to some arbitrary criteria. The user can achieve these goals by setting up a functional-encryption system, then giving the proxy a key sk[*f*] where *f* is the user-specified program that outputs 1 if the plaintext is spam and 0 otherwise. The proxy can use sk[*f*] to test if an encrypted message is spam without learning anything more about the plaintext (see the figure here).

One can naturally consider generalizations of this idea; for instance, the proxy might selectively send important email messages (as deemed by the function *f*) to the user's mobile device. Taking things further we can imagine the destination of a packet is encrypted, and the secret key sk[*f*] allows a router to learn the next hop and nothing more.

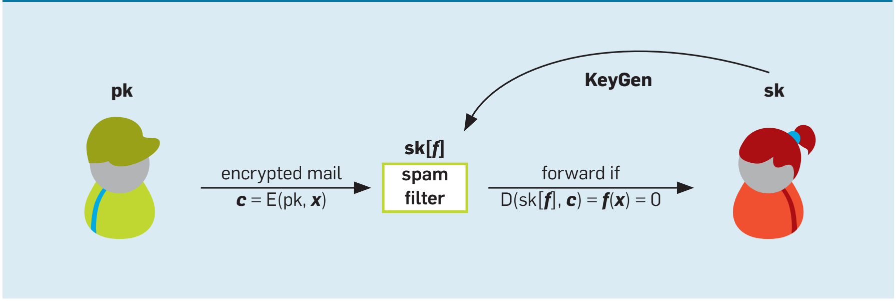
Expressive access control. In large organizations a user will often think of sharing data according to some access policy. In addition to our corporate example, this might also occur in other domains (such as health care, insurance companies, government institutions, and universities). Bridging the gap between how a user thinks of sharing data and discovering the public keys of all other users who match or will match such sharing can be difficult and is subject to the problems outlined earlier; for example, a system might try to encrypt data separately to the public key of every user matching a certain policy. However, as also noted, this user-specific encryption requires identification of each user, as well as the overhead of encrypting to each one individually. Moreover, this user-specific encryption does not cover users who do not meet the criteria today but will in the future.

Using functional encryption a user can directly express how the user (or organization) wishes to share the data in the encryption process. In particular, the user can encrypt $x = (P,m)$ where *m* is the data the user wishes to share, and *P* is the access policy that describes how the user wants to share it. The user's secret-key function sk[*f*] will then check whether the user's credentials or attributes match the policy and reveal only *m* in this case. Corresponding to the example of an accounting or sales department with an office in the company's main building, *P* could embed the policy ("ACCOUNTING" OR "SALES") AND "MAIN BUILDING." A recipient's function *f* would embed the attributes of the particular user and check if they satisfy the formula and if so return *m*.

Mining large datasets. Data mining is used in medical research, social networks, network security, and financial fraud detection. Administrators often want to give users the ability to mine datasets for certain types of queries but not let them learn anything else. Consider a medical researcher who wants to test if there is a link between a genotype and a type of cancer in a particular ethnic group. If the administrator has data consisting of patient gene sequences and medical history, the administrator would like to give the researcher the ability to test for this linkage, without revealing the details of all patients' medical conditions.

Note that in practice, an administrator typically does not know the queries that will be of interest until well after the data is created and stored. Functional encryption provides an elegant solution. When data is created it is simply encrypted in a functional-encryption system. Later, a user requests to be allowed to learn a certain query or function *f* of the data. If data access is authorized, the user is given sk[*f*] and can apply this key to (attempt to) decrypt existing or future encrypted data. Thus, in a functional-encryption system supporting a class of functions **F** a user could be given the ability to

The email recipient, who has a master secret key sk, gives a spam-filtering service a key sk[*f*] for the functionality *f*; this *f* satisfies *f*(*x*) = 1 whenever message *x* is marked as spam by a specific spam predicate, otherwise *f*(*x*) = 0. A sender encrypts an email message *x* to the recipient, but the spam filter blocks the message if it is spam. The spam filter learns nothing else about the contents of the message.



compute any function from this class on the dataset.

These three examples of functionality motivate the research agenda we put forward here—to create functional-encryption systems supporting the richest possible families of functions and understand what limitations are inherent for functional-encryption systems.

Functional Encryption

Recall that public-key encryption systems (such as RSA and El-Gamal) consist of three algorithms:

Setup. Outputs a secret key denoted sk and a public key pk; anyone can encrypt message using pk, but only the secret key holder is able to decrypt;

Encryption E. Takes a public key pk and a message as input and outputs a ciphertext; and

Decryption D. Takes a secret key sk and a ciphertext as input and outputs a message.

A functional-encryption system includes the same three algorithms but also includes a fourth algorithm called KeyGen. Here, the secret key output by the Setup algorithm is called the master key, denoted by mk. The KeyGen algorithm takes as input mk and the description of some function f. It outputs a key that is specific to the function f and denoted sk[f]. More precisely, if c is the result of encrypting data x with public key pk, then

$$D(\mathrm{sk}[f]; c) \text{ outputs } f(x)$$

We emphasize that sk[f] does not fully decrypt c, outputting only a function f of the full decryption. To fully decrypt c the authorized user can use a secret key sk[g], where g is the identity function, namely $g(x) = x$ for all x.

Informally, security of a functional-encryption system means an attacker with a set of secret keys $\mathrm{sk}[f_1],\ldots,\mathrm{sk}[f_\ell]$ can learn nothing about the decryption of some ciphertext c other than what is revealed by the keys at the attacker's disposal.

To illustrate the power of functional encryption, the following section covers how it naturally captures many advanced encryption concepts in cryptography. First, it should be clear that traditional public-key encryption is a very special case of functional encryption, where the only supported function is the identity function; the decryptor learns either the complete decryption or nothing at all.

A filtering server can use the user's functional secret key to test if an encrypted email message is spam without learning anything else about the plaintext.

Identity-based encryption. A more advanced public-key concept called "identity-based encryption," or IBE, is an encryption system where any string can serve as a public key; a user's email address, a date, an IP address, a location, or even the numbers 1, 2, and 3 are all potential public keys. IBE public keys are often called "identities" and denoted by id. To obtain the secret key for a particular identity the user communicates with an authority holding a master key. The authority verifies the user is authorized to receive the requested secret key, and, if so, it generates the secret key using its master key. IBE was proposed by Shamir[29] in 1984, and the first implementations of IBE were proposed by Boneh and Franklin[6] and Cocks[10] in 2001; notable constructions include Agrawal et al.,[1] Boneh and Boyen,[4] Gentry,[15] Gentry et al.,[16] Waters,[30] and Waters.[31]

Using the terminology of functional encryption the IBE problem can be recast as an equality testing functionality. Let pk and mk be the output of the functional encryption setup algorithm. To encrypt a message m to identity id the encryptor calls the encryption algorithm as

$$E(\mathrm{pk}; (\mathsf{id}, m))$$

and obtains a ciphertext c. Note the data being encrypted is the pair (id, m).[a] A recipient with identity id * obtains a secret key for id * by asking the authority for a secret key $\mathrm{sk}[f_{\mathsf{id}^*}]$ where the function f_{id^*} is defined[b] as

$$f_{\mathsf{id}^*}((\mathsf{id}, m)) := \begin{cases} m \text{ if } \mathsf{id} = \mathsf{id}^*, \\ \perp \text{ otherwise} \end{cases}$$

The authority generates $\mathrm{sk}[f_{\mathsf{id}^*}]$ using its functional master key mk. Using this secret key the user can decrypt messages intended for identity id* but learns nothing about messages encrypted for other identities.

Recall that IBE systems reduce reliance on certificate directories needed

a Using our earlier notation we would have $x = (\mathsf{id}, m)$, though we omit x for readability.

b We use the $\perp$ symbol as a special symbol to denote failure to decrypt.

for traditional public-key encryption; to encrypt to identity id, the encryptor needs only the global public key pk and the recipient's identity id. General functional encryption systems have the same property: they require no online certificate directory. An encryptor needs only the global public key pk and the payload x to be encrypted and no other information about the intended recipient(s).

Attribute-based encryption. Another encryption concept called "attribute-based encryption," or ABE, lets the encryptor specify more abstractly who is authorized to decrypt a specific ciphertext. ABE was proposed by Sahai and Waters[28] and later refined by Goyal et al.[19] into two different formulations: key-policy ABE and ciphertext-policy ABE.

In the ciphertext-policy system the encryptor specifies a policy φ on recipient attributes that determines who can decrypt the ciphertext; for example, the encryptor can encrypt messages to anyone who is a

("U.S. citizen" and "female")
or ("over 30")

which is a Boolean formula φ on three variables. To encrypt a message m with decryption policy φ the encryptor calls

$$E(\text{pk}, (\varphi, m))$$

and obtains a ciphertext c.

Now, consider a recipient who wants to decrypt the ciphertext. The recipient has a number of attributes, say,

"U.S. citizen," "Rhodes Scholar,"
"female," "under 30"

Let n be the total number of attributes, and we represent the set of user attributes as a Boolean vector of length n; the vector is 1 at positions that correspond to attributes that are true and 0 everywhere else. With this setup each user has an attribute vector $\mathbf{u}$ in $\{0, 1\}^n$.

A recipient with attribute vector $\mathbf{u}$ obtains a secret key for his attribute vector by asking the authority for a secret key sk$[f_\mathbf{u}]$ where the function $f_\mathbf{u}$ is defined as

$$f_\mathbf{u}((\varphi, m)) := \begin{cases} m \text{ if } \varphi(\mathbf{u}) = 1, \\ \perp \text{ otherwise} \end{cases}$$

A tantalizing question is whether techniques from lattices, which have been so useful in the context of fully homomorphic encryption, can help achieve greater functionality for functional encryption.

The authority generates sk$[f_\mathbf{u}]$ using its functional master key mk. Using this secret key the user can decrypt ciphertexts where the user's attributes satisfy the decryption policy but learns nothing about the decryption of other ciphertexts.

A related concept called "key-policy attribute-based encryption" places the access policy φ in the key and the vector $\mathbf{u} \in \{0, 1\}^n$ in the ciphertext. The secret key sk$[f_\varphi]$ decrypts all encryptions $E(\text{pk}, (u, m))$ for which $\varphi(\mathbf{u}) = 1$.

Security

Here we turn to constructing functional-encryption systems but first explain what it means for a functional system to be secure. The full definition is a bit technical, and we give only high-level intuition; for more, see Boneh et al.[7]

Roughly speaking, a functional-encryption system is secure if an attacker with a set of secret keys sk$[f_1]$,…,sk$[f_t]$ can learn nothing about the decryption of some ciphertext c other than what is revealed by the keys at the attacker's disposal. If c is the encryption of some data x, then the attacker can use the attacker's secret keys to learn $f_1(x)$,…,$f_t(x)$. However, the attacker must be unable to learn anything else about x; for example, if the attacker has secret keys that reveal the first three bits of x, then clearly the attacker can learn these bits, given an encryption of x but would be unable to learn anything about the remaining bits of x.

To give a bit more detail about security requirements, let A be a polynomial-time adversary that takes as input three things: the public key pk, a set of secret keys sk$[f_1]$,…,sk$[f_t]$ for functions f_1,…, f_t of its choice, and a ciphertext $c = E(\text{pk}, x)$. This A might output some information about the decryption of c (such as the least significant bit of x). We say the system is secure if for every such A there is a another polynomial-time algorithm B, called a simulator, that, given pk and $f_1(x)$,…, $f_t(x)$ but not given c is able to output the same information about x that A output. Since B never got to see c it must have deduced the information about x strictly from $f_1(x)$,…, $f_t(x)$. Since A and B output the same information about x, the existence of B means the only information A can learn about x from the ciphertext c is information it can learn from $f_1(x),\ldots, f_t(x)$ but cannot learn anything else about x. Hence, A can learn from c whatever is revealed by the secret keys at its disposal but nothing else.[c]

Challenge: Preventing collusion attacks. Attacks on functional encryption using multiple functional secret keys are called collusion attacks, and preventing them is the main obstacle to constructing secure functional systems. To illustrate the problem consider again the functionality described earlier and suppose the encryptor wishes to encrypt a message m to this policy

"U.S. citizen" and "over 30"

A simple implementation is to associate a public key pk_1 with the attribute "U.S. citizen" and a public key pk_2 with the attribute "over 30" and double-encrypt the message m as

$$c = E(\mathrm{pk}_1, E(\mathrm{pk}_2, m))$$

where here $E(\cdot,\cdot)$ is a regular public-key-encryption algorithm. To decrypt c the recipient—call her Alice—must possess both secret keys corresponding to pk_1 and pk_2, implementing the conjunction policy specified by the encryptor.

Now, suppose another user, Bob, has attributes "U.S. citizen" and "male" where the attribute "male" is associated with a public key pk_3. He would be given the secret keys corresponding to pk_1 and pk_3, letting him decrypt messages encrypted for some policies (such as ("U.S. citizen" and "male")). In addition, suppose Alice has the attribute "over 30" and is then given only the secret key corresponding to pk_2. Thus, she cannot decrypt the message associated with the original policy on her own.

The problem is Alice and Bob can collude to combine their secret keys and create new secret keys neither one should have; for example, Alice and Bob working together can decrypt ciphertexts intended for policy "over 30" and "male," even though neither can decrypt the ciphertext by themselves. In this example, collusion enabled Alice and Bob to decrypt a ciphertext to which neither should have access.

Secure constructions. Secure constructions for complex functionalities must prevent collusion attacks. Collusion attacks are prevented by binding together all secret keys for a set of attributes, so mixing the keys given to distinct users does not help. As a visual metaphor, one can imagine that all the keys given to Alice are colored blue, while all the keys given to Bob are colored red. Decryption succeeds only when the decryptor uses a set of keys of the same color. The colors ensure Alice and Bob cannot combine their keys to decrypt ciphertexts they should not be able to decrypt.

In practical terms, the colors are implemented through randomization values. All the keys given to Alice are blinded by the same random value, while all the keys given to Bob are blinded by a different random value. Decryption with keys blinded by the same randomizer produces the correct decrypted message. Decryption with keys blinded by different randomizers results in a random value unrelated to the correct decryption.

State of the Art

The state of the art in functional encryption can be understood by considering what information about the plaintext x is exposed by the ciphertext to all participants. We refer to this information as the result of the "empty functionality" denoted $f_\varepsilon(\cdot)$; for example, it is inherent in any encryption scheme that the empty functionality exposes some information about x (such as a bound on the size of the plaintext). When the exact plaintext length is leaked by the ciphertext we write $f_\varepsilon(x) = |x|$ to indicate that anyone can learn the plaintext length from the ciphertext.

Public index: ABE. In general, we can consider the problem of functional encryption where the data to be encrypted is decomposed into two parts $x = (\mathsf{ind}, m)$, where ind denotes a public index the encryptor does not mind revealing to all participants in the system. That is, we define the empty functionality as $f_\varepsilon(\mathsf{ind}, m) = (\mathsf{ind}, |m|)$.

Now consider the specific case of ABE, where the access policy φ is now considered a public index. In it, where access policy does not require protection, we have fairly broad and efficient constructions of secure ABE schemes; secure ABE schemes exist that support any access policy φ that can be expressed as a Boolean formula over the attributes (as in the earlier examples).[3,19,22,23,25,26,32] Going beyond policies expressible as Boolean formulas remains a vexing open problem for researchers, with the ultimate goal of supporting policies expressible as arbitrary Boolean circuits or Turing Machines.

Non-public index. A more challenging setting arises where we insist the empty functionality reveals as little as possible, namely $f_\varepsilon(x) = |x|$. Here, our current understanding of functional encryption is extremely limited. The state of the art is limited to the inner-product functionality over prime fields.[2,21,22,25] Because this functionality is somewhat technical (and before we describe it more formally), we briefly discuss some applications: First, consider the question of searching on encrypted data, where the data is encrypted based on a public key and stored on a public server.[5] The security challenge in this setting is to hide the specific nature of the search query from the public server while still allowing the public server to send back only data entries that match the search query. The inner-product functionality we describe in the following paragraphs allows a user to perform such a search based on disjunction queries and more generally searches defined by CNF and DNF formulae or by checking whether a univariate search polynomial evaluates to zero on a particular input value.

The functionality we consider is defined over a prime field F_p where p is a large prime chosen randomly during the setup of the functional-encryption scheme. Messages and keys will correspond to vectors of a fixed arbitrary dimension n over F_p. Let us denote the message by the vector $\mathbf{v}$ and the vector underlying a secret key by $\mathbf{u}$. We then have

$$f_{\mathbf{u}}(\mathbf{v}) := \begin{cases} 1 \text{ if} \sum_{i=1,\ldots,n} \mathbf{u}_i \cdot \mathbf{v}_i = 0, \\ \perp \text{ otherwise} \end{cases}$$

To see how this functionality can be applied, consider again the example of disjunction queries: Suppose a ciphertext is meant to encrypt a single keyword we hash down to a value a in our finite field F_p. Then to encrypt this val-

c Note our security model does not rely on any assumption about trusted hardware or online servers needed during decryption.

ue a, the system actually encrypts the vector $\mathbf{v} = (1, a, a^2, \ldots, a^{n-1})$. Now, suppose we have to create a key corresponding to a disjunction query "a_1 OR a_2 OR a_3". We do this by first considering the polynomial $p(x) = (x - a_1)(x - a_2)(x - a_3)$, writing it out in standard form as $p(x) = c_0 + c_1x + c_2x^2 + c_3x^3$, where the c_i are the appropriate coefficients. We then issue a key for the vector $\mathbf{u} = (c_0, c_1, c_2, c_3, 0, \ldots, 0)$. Glancing at the functionality, we see our key will indeed match the ciphertext for value a if and only if $p(a) = 0$; that is, if the value a is a root of our polynomial $p(x)$, which was designed to have roots only at the three values a_1, a_2, a_3 in our desired disjunction. Other special cases of inner products, including conjunctions and range testing functionalities, were considered in Boneh and Waters.[8]

Unfortunately, the exact cryptographic mechanisms by which the results work in Katz et al.,[21] Lewko et al.,[22] and Okamoto and Takashima[25] are too technically involved to describe here; we encourage all to look into these sources for further technical detail.

Current limitations. Current functional-encryption schemes, especially in non-public index settings, are limited. From a technical standpoint, current techniques for building functional-encryption schemes are all based on elliptic-curve groups equipped with efficiently computable bilinear pairings that map into the multiplicative structure of a finite field. At a very high level of design abstraction a pairing operation allows for a single multiplication between the exponents of two "source" group elements. However, the result of a pairing operation is a "target" group for which the operation cannot be repeated.

The reason we can handle inner products of two vectors is because this operation requires only one parallel call to the multiplication operation, which is all that bilinear maps provide. A tantalizing question is whether techniques from lattices, which have been so useful in the context of fully homomorphic encryption,[14] can help achieve greater functionality for functional encryption.

Efficiency. The efficiency of functional-encryption systems varies significantly with specific cryptographic constructions. However, we can offer an approximate sense of the efficiency of the ABE where the ciphertext is associated with any access policy φ that can be expressed as a Boolean formula over attributes. In current systems, the size of the ciphertext scales with the size of the Boolean formula φ; for example, in Waters,[32] a ciphertext consisted of two group elements for every leaf node of φ, and encryption took three exponentiations for every leaf node. Decryption requires two of the aforementioned pairing operations for each attribute used in the formula. While difficult to predict how future functional-encryption systems might evolve, developers could expect that the number of public-key operations required will scale with the complexity of the functionality.

Functional Encryption vs. Fully Homomorphic Encryption

Fully homomorphic encryption (FHE) is arguably the most impressive development in cryptography over the past few years, enabling one to compute on ciphertexts in the following sense: Given a public key pk, encryptions of messages $x_1, \ldots, x_t$ under pk, and the description of a function f as input, any user can construct an encryption of the message $f(x_1, \ldots, x_t)$; see Gentry[13] for a detailed discussion. A more restricted version of FHE, called univariate FHE, allows any user to construct an encryption of $f(x)$ from an encryption of x for all univariate functions f.

While both FHE and functional encryption support some form of computation on ciphertexts, it is not known how to construct functional encryption from FHE; FHE does not even seem to imply basic functionalities (such as identity-based encryption). The reason for this limitation is that the output of an FHE computation on encrypted data is an encrypted result; in contrast, the output of a functional-encryption computation is available in the clear.

To further illustrate the difference between FHE and functional encryption recall the spam-filtering example discussed at the beginning of the article. In it, the spam filter was given a secret key sk = sk[f], where f is a function that outputs 1 if an email is spam and 0 otherwise. The key sk lets the spam filter run the spam predicate f on encrypted email messages and block encrypted spam. With FHE, the spam filter can likewise run the spam predicate f on encrypted email messages, but the filter learns only the encrypted output of the predicate; it does not and cannot learn whether an encrypted email message is spam. In particular, with FHE, the filter can tag an encrypted email message with only an encrypted tag indicating "spam" or "not spam" but cannot block spam email messages for the end user. This example illustrates the potential power of functional encryption over FHE. However, constructing a fully functional encryption scheme is still an open problem, whereas FHE constructions exist.

Generalizations

Here, we cover a few generalizations, variants, and extensions of functional encryption that are motivated in practice:

Delegating keys. Users might sometimes want to delegate a limited set of their capabilities to another user or device; for example, a medical researcher with a secret key able to decrypt raw medical records might want to distribute to a grad student a key that can output only certain statistics (such as averages over the data). As another example, suppose users are planning to travel with their mobile devices but are concerned the devices might be lost or stolen; they might then want to copy a key to the devices that decrypts only the data that was encrypted during the travel time or restrict the key to capabilities related only to the purpose of the trip.

A simple approach is for users with a key sk[f] to query the authority for a more restrictive key sk[f'] anytime they wish to delegate a key for a more restrictive function f'. However, involving the authority in every delegation is cumbersome, exposes an online authority to more risk, and will not work if the authority is unreachable. Therefore, we would like the delegation operation to be autonomous. Roughly, a user with sk[f] can create sk[f'] if f' is more limited than the function f; whatever we can learn from f' we can learn from f.

The concept of delegation arose in identity-based encryption in Gentry and Silverberg[17] and in Horwitz and

Lynn[17,20] and can be realized in attribute-based encryption.[19]

Functionality over multiple authorities. In a standard functional-encryption system, one authority is responsible for issuing private keys, though some systems might require more flexibility. Returning to the example of (ciphertext-policy) attribute-based encryption, in a standard system, one authority is responsible for both determining what attributes/credentials to issue to each user and creating the keys.

While a single-authority solution is likely workable for smaller organizations, in many applications a user might want to create policies spanning many trust domains; for instance, suppose we wish to encrypt a document for all military personnel who are also members of the ACM, asking who should manage the system? Using a central authority creates several problems; for one, no single party is always able to speak authoritatively for multiple trust domains or organizations. Indeed, a user might wish to create a policy that spans organizations that are not even aware of one another. Another core limitation is that a central authority creates a central performance bottleneck and consolidates trust in one entity. Are any two different organizations able to agree who to trust in this role?

Recent work in decentralized attribute-based encryption[9,24] has sought to overcome these limitations so users are able to encrypt according to an ABE policy issued as a formula over attributes issued from different authorities. An interesting direction is to see what other functionalities beyond ABE might arise from the use of multiple authorities in functional-encryption systems.

Functional encryption with public-key infrastructure. Finally, we consider how ideas from functional encryption can be applied to other scenarios; specifically, consider a scenario involving the following conditions:

Per-user infrastructure. There exists a per-user public-key infrastructure where every user u obtains a secret key $\mathrm{sk_u}[f]$ for some function f_u appropriately chosen by the user and also establishes a public key pk_u unique to the user; this public key should also not leak information about the function f_u. Such a key is established through interaction between an authority and the user;

Targeting a specific key. Encryptions are always targeted at a specific user's public key $\mathrm{pk_u}$. However, the encryptor does not know the function f_u corresponding to the user hidden by the public key $\mathrm{pk_u}$. At the same time, if a user u obtains an encryption of x under the user's public key pk_u, then decryption allows the user to learn $f_u(x)$ but nothing else. Users should also not be able to obtain additional capabilities by combining secret keys corresponding to different public keys; and

Misbehaving central authority. A misbehaving central authority must not be able to decrypt encryptions intended for honest users in the system.

We stress this scenario is quite different from the functional-encryption scenario we have considered here. One of the key properties of functional encryption is it does not require public-key directories, thus enabling a variety of applications (such as secure storage in the cloud and secure searching on encrypted data). At the same time, the support comes at the cost of users needing to trust a key-generation authority (or set of such authorities) capable of breaking the security of ciphertexts.

This scenario was considered in recent work[27] where it was shown that in this setting, called "worry-free encryption," the system can support functions (in the non-public index setting) specified by any arbitrary polynomial-size circuit—significantly beyond what is possible with standard functional encryption today. However, it must be stressed that this setting does not cover motivating applications of functional encryption (such as secure storage in the cloud and searching on encrypted data); see Sahai and Seyalioglu[27] for more detail on this setting.

While existing functional-encryption systems are remarkably expressive, the central challenge is to construct a system that supports creation of keys for any function in both public and non-public index settings.

Future of Functional Encryption

What will functional encryption look like in 10 years? While existing functional-encryption systems are remarkably expressive, the central challenge is to construct a system that supports creation of keys for any function in both public and non-public index settings. If we could create such systems we could imagine embedding anything from arbitrarily complex spam filters to image-recognition algorithms into encryption systems. Imagine an encryption system that lets a user view only an image if a facial-recognition algorithm matches a picture of the user to a face in the encrypted image. Moreover, the output of the decryption could show the area immediately surrounding the identified user and blur out the rest of the image.

Current progress on building functional encryption systems is dominated by the tool of groups with bilinear maps mentioned earlier. However, as also mentioned earlier there are reasons to suspect there might be fundamental barriers to realizing more advanced functional encryption systems from this tool.

Cryptography researchers need to search further out, though a reason for optimism is the recent dramatic leap in what we can achieve in homomorphic encryption systems. Hopefully, such a leap will be achieved in the not-too-distant future (perhaps using related techniques) in the realm of functional encryption.

Finally, more applied research is needed to build functional encryption into real-world systems, as well as to specify formats for attribute spaces and languages for expressing access policies. Due to the expressive power of these systems we hope to see real-world deployments of functional encryption over the next decade. The end result is much greater flexibility in specifying who can and cannot access protected data.

Acknowledgments
Dan Boneh is supported by the National Science Foundation, the Defense Advanced Projects Agency PROgramming Computation on EncryptEd Data (PROCEED) program, the Air Force Office of Scientific Research under the Multidisciplinary University Research Initiative award for "Collaborative policies and assured information sharing" (Project PRESIDIO), a Google Faculty Research Award, Samsung, and the Packard Foundation.

Amit Sahai is supported by a Defense Advanced Research Projects Agency/Office of Naval Research PROCEED award, NSF grants 1228984, 1136174, 1118096, 1065276, 0916574, and 0830803, a Xerox Foundation Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based on work supported by DARPA through the U.S. Office of Naval Research under Contract N00014-11-1-0389.

Brent Waters is supported by NSF CNS-0915361 and CNS-0952692, Air Force Office of Scientific Research Grant No: FA9550-08-1-0352, DARPA PROCEED, DARPA N11AP20006, Google Faculty Research award, an Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and the Packard Foundation.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the U.S. Department of Defense or the U.S. government. C

References
1. Agrawal, S., Boneh, D., and Boyen, X. Efficient lattice (H)IBE in the standard model. In *Proceedings of EUROCRYPT, Lecture Notes in Computer Science*, Springer, 2010, 553–572.
2. Agrawal, S., Freeman, D.M., and Vaikuntanathan, V. Functional encryption for inner product predicates from learning with errors. In *Proceedings of ASIACRYPT, Lecture Notes in Computer Science*, Springer, 2011, 21–40.
3. Bethencourt, J., Sahai, A., and Waters, B. Ciphertext-policy attribute-based encryption. In *Proceedings of the IEEE Symposium on Security and Privacy*, IEEE Computer Society, 2007, 321–334.
4. Boneh, D. and Boyen, X. Efficient selective-id secure identity-based encryption without random oracles. In *Proceedings of EUROCRYPT, Lecture Notes in Computer Science*, Springer, 2004, 223–238.
5. Boneh, D., Crescenzo, G.D., Ostrovsky, R., and Persiano, G. Public-key encryption with keyword search. In *Proceedings of EUROCRYPT, Lecture Notes in Computer Science*, Springer, 2004, 506–522.
6. Boneh, D. and Franklin, M.K. Identity-based encryption from the weil pairing. In *Proceedings of Crypto, Lecture Notes in Computer Science*, Springer, 2001, 213–229.
7. Boneh, D., Sahai, A., and Waters, B. Functional encryption: Definitions and challenges. In Proceedings of TCC, Lecture Notes in Computer Science, Springer, 2011, 253–273.
8. Boneh, D. and Waters, B. Conjunctive, subset, and range queries on encrypted data. In *Proceedings of TCC, Lecture Notes in Computer Science*, Springer, 2007, 535–554.
9. Chase, M. Multi-authority attribute-based encryption. In *Proceedings of TCC, Lecture Notes in Computer Science*, Springer, 2007, 515–534.
10. Cocks, C. An identity-based encryption scheme based on quadratic residues. In *Proceedings of the Institute of Mathematics and Its Applications, Lecture Notes in Computer Science*, Springer, 2001, 360–363.
11. Diffie, W. and Hellman, M.E. Multiuser cryptographic techniques. In *Proceedings of AFIPS National Computer Conference*, AFIPS Press, 1976, 109–112.
12. Diffie, W and Hellman, M.E. New directions in cryptography. *IEEE Transactions on Information Theory 22* (1976), 644–654.
13. Gentry, C. Computing arbitrary functions of encrypted data. *Commun. ACM 53*, 3 (Mar. 2010), 97–105.
14. Gentry, C. Fully homomorphic encryption using ideal lattices. In *Proceedings of STOC 2009*, ACM Press, New York, 2009, 169–178.
15. Gentry, C. Practical identity-based encryption without random oracles. In *Proceedings of EUROCRYPT, Lecture Notes in Computer Science*, Springer, 2006, 445–464.
16. Gentry, C., Peikert, C., and Vaikuntanathan, V. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of STOC*, ACM Press, New York, 2008, 197–206.
17. Gentry, C. and Silverberg, A. Hierarchical id-based cryptography. In *Proceedings of ASIACRYPT 2002, Lecture Notes in Computer Science*, Springer, 2002, 548–566.
18. Goldreich, O., Micali, S., and Wigderson, A. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of STOC*, ACM Press, New York, 1987, 218–229.
19. Goyal, V., Pandey, O., Sahai, A., and Waters, B. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the ACM Conference on Computer and Communications Security*, ACM Press, New York, 2006, 89–98.
20. Horwitz, J. and Lynn, B. Toward hierarchical identity-based encryption. In *Proceedings of EUROCRYPT, Lecture Notes in Computer Science*, Springer, 2002, 466–481.
21. Katz, J., Sahai, A., and Waters, B. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of EUROCRYPT, Lecture Notes in Computer Science*, Springer, 2008, 146–162.
22. Lewko, A.B., Okamoto, T., Sahai, A., Takashima, K., and Waters, B. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Proceedings of EUROCRYPT, Lecture Notes in Computer Science*, Springer, 2010, 62–91.
23. Lewko, A.B., Sahai, A., and Waters, B. Revocation systems with very small private keys. In *Proceedings of the IEEE Symposium on Security*, IEEE Computer Society, 2010, 273–285.
24. Lewko, A.B. and Waters, B. Decentralizing attribute-based encryption. In *Proceedings of EUROCRYPT, Lecture Notes in Computer Science*, Springer, 2011, 568–588.
25. Okamoto, T. and Takashima, K. Fully secure functional encryption with general relations from the decisional linear assumption. In *Proceedings of CRYPTO, Lecture Notes in Computer Science*, Springer, 2010, 191–208.
26. Ostrovsky, R., Sahai, A., and Waters, B. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the ACM Conference on Computer and Communications Security*, ACM Press, New York, 2007, 195–203.
27. Sahai, A. and Seyalioglu, H. Worry-free encryption: Functional encryption with public keys. In *Proceedings of the ACM Conference on Computer and Communications*, ACM Press, New York, 2010, 463–472.
28. Sahai, A. and Waters, B. Fuzzy identity-based encryption. In *Proceedings of EUROCRYPT, Lecture Notes in Computer Science*, Springer, 2005, 457–473.
29. Shamir, A. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO, Lecture Notes in Computer Science*, Springer, 1984, 47–53.
30. Waters, B. Efficient identity-based encryption without random oracles. In *Proceedings of EUROCRYPT, Lecture Notes in Computer Science*, Springer, 2005, 114–127.
31. Waters, B. Dual-system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *Proceedings of CRYPTO, Lecture Notes in Computer Science*, Springer, 2009, 619–636.
32. Waters, B. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *Proceedings of the Public Key Cryptography Conference, Lecture Notes in Computer Science*, Springer, 2011, 53–70.
33. Yao, A.C.-C. Protocols for secure computations (extended abstract). In *Proceedings of FOCS*, IEEE Computer Society, 1982, 160–164.

Dan Boneh (dabo@cs.stanford.edu) is a professor of computer science and electrical engineering at Stanford University.

Amit Sahai (sahai@cs.ucla.edu) is a professor of computer science at the University of California, Los Angeles.

Brent Waters (bwaters@cs.utexas.edu) is an assistant professor of computer science at the University of Texas at Austin.