

UNITED STATES PATENT AND TRADEMARK OFFICE

---

BEFORE THE PATENT TRIAL AND APPEAL BOARD

---

FACEBOOK, INC., INSTAGRAM, LLC and WHATSAPP INC.,  
Petitioners

v.

BLACKBERRY LIMITED  
Patent Owner

---

U.S. Patent No. 7,372,961  
Issue Date: May 13, 2008

Title: METHOD OF PUBLIC KEY GENERATION

---

**DECLARATION OF JONATHAN KATZ, PH.D.**

Facebook's Exhibit No. 1002  
0001

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

I.	INTRODUCTION AND QUALIFICATIONS .....	5
A.	Qualifications and Experience .....	5
B.	Materials Considered.....	7
II.	PERSON HAVING ORDINARY SKILL IN THE ART .....	10
III.	<b>LEGAL PRINCIPLES USED IN THE ANALYSIS</b> .....	13
B.	Prior Art.....	13
C.	Claim Construction.....	14
D.	Legal Standards for Obviousness .....	16
IV.	<b>RELEVANT TECHNOLOGY BACKGROUND</b> .....	21
B.	Random Number Generation .....	22
C.	Cryptography and Its Reliance on Random Number Generators .....	23
D.	Modular Reduction and the Modulo Bias Problem .....	29
E.	Rejecting Random Numbers Above a Desired Range, aka Rejection Sampling .....	34
V.	<b>OVERVIEW OF THE '961 PATENT</b> .....	46
A.	The Specification.....	46
B.	The Challenged Claims .....	51
C.	Interpretation of Claim Limitations in the '961 Patent.....	53
VI.	<b>APPLICATION OF THE PRIOR ART TO THE CHALLENGED CLAIMS OF THE '961 PATENT</b> .....	59
B.	Brief Description and Summary of the Prior Art.....	60
1.	Brief Summary of DSS (EX. 1004) .....	60
2.	Brief Summary of Rose (EX. 1006) .....	66
3.	Brief Summary of Menezes (EX. 1005) .....	70
4.	Brief Summary of Schneier (EX. 1008) .....	72
5.	Brief Summary of Rao (EX. 1018) and Floyd (EX. 1019).....	73

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

C.	Ground 1: Comparison of the Prior Art to the Challenged Claims Based on DSS, Schneier, and Rose.....	75
1.	Independent Claim 1 .....	75
2.	Dependent Claim 2: The method of claim 1 wherein another seed value is generated by said random number generator if said output is rejected.....	116
3.	Dependent Claim 5: The method of claim 1 wherein said order q is a prime number represented by a bit string of predetermined length L. ....	116
4.	Independent Claim 15:.....	117
5.	Dependent Claim 16: The computer readable medium of claim 15 wherein another seed value is generated by said random number generator if said output is rejected.....	120
6.	Dependent Claim 19: The computer readable medium of claim 15 wherein said order q is a prime number represented by a bit string of predetermined length L.....	121
D.	Ground 2: Comparison of the Prior Art to the Challenged Claims Based on DSS, Schneier, and Menezes .....	121
2.	Independent Claim 1 .....	122
3.	Dependent Claim 2: The method of claim 1 wherein another seed value is generated by said random number generator if said output is rejected.....	135
4.	Dependent Claim 5: The method of claim 1 wherein said order q is a prime number represented by a bit string of predetermined length L. ....	136
5.	Independent Claim 15:.....	136
6.	Dependent Claims 16, 19.....	136
E.	Grounds 3-4: Comparison of the Prior Art from Grounds 1-2, in Further View of Rao and Floyd.....	137
2.	Independent Claim 23: .....	137

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

3. Dependent Claim 24: The cryptographic unit of claim  
23 wherein another seed value is generated by said  
random number generator if said output is rejected. ....145
4. Dependent Claim 27: The cryptographic unit of claim  
23 wherein said order  $q$  is a prime number represented by  
a bit string of predetermined length  $L$ . ....146

VII. CONCLUSION 146

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

I, Jonathan Katz, PhD., declare as follows:

1. I have personal knowledge of the facts stated in this declaration, and could and would testify to these facts under oath if called upon to do so.

**I. INTRODUCTION AND QUALIFICATIONS**

**A. Qualifications and Experience**

2. I received an undergraduate degree in mathematics from the Massachusetts Institute of Technology in 1996. My coursework included several classes in computer science. I entered the Ph.D. program in computer science at Columbia University in 1998. My doctoral work focused on cryptography.

3. In early 2000, I taught a course “Introduction to Programming in C” at Columbia University, and in early 2001, I taught a graduate-level “Introduction to Cryptography” course there. In 2002, I received my Ph.D. (with distinction) in computer science; my doctoral thesis was entitled “Efficient Cryptographic Protocols Preventing ‘Man-in-the-Middle’ Attacks.”

4. From May 1999 to March 2000, while I was conducting my doctoral research, I worked as a security consultant for Counterpane Systems. In connection with my work for Counterpane, I discovered security flaws in the popular email encryption software Pretty Good Privacy (PGP). This work was covered in the press

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

and led to two published papers and a refinement of existing standards for email encryption. I also designed and implemented secure cryptographic protocols for clients. In March 2000, I began work as a Research Scientist in the cryptography group at the Mathematical Sciences Research Center of Telcordia Technologies.

5. Since 2002, I have worked as a professor in the Department of Computer Science at the University of Maryland. In that capacity I regularly teach courses in cryptography and cybersecurity at both the undergraduate and graduate level, conduct research in those fields, and supervise graduate-student research. I was appointed Director of the Maryland Cybersecurity Center in 2013. In that capacity I regularly interact with cybersecurity researchers at other academic institutions around the world, as well as with industry professionals working in the area of cybersecurity.

6. In 2007, I co-authored a textbook *Introduction to Modern Cryptography*, now in its second edition. This text has been used in courses on cryptography worldwide. I also authored the book *Digital Signatures* in 2010. Since 2011, I have served as an editor of the Journal of Cryptology, the leading journal in the field. In 2016 and 2017, I served as co-program chair of the annual Crypto conference, the leading and longest-running academic conference focused on

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

cryptography. I am currently serving as co-program chair of the ACM Conference on Computer and Communications Security, one of the top conferences in the field of cybersecurity. A list of other program committees and editorial boards I have served on can be found as part of my curriculum vitae.

7. I have attached a more detailed list of my qualifications as **Exhibit A**.

8. I am being compensated for my time working on this matter at my standard hourly rate plus expenses. I have no personal or financial stake or interest in the outcome of the present proceeding, and the compensation is not dependent on the outcome of this IPR and in no way affects the substance of my statements in this Declaration.

**B. Materials Considered**

9. The analysis that I provide in this Declaration is based on my education and experience in cryptography, software, and mathematics, as well as the documents I have considered, including U.S. Patent No. 7,372,961 (the “’961 Patent”) [EX1001], which states on its face that it issued from an application filed on December 26, 2001, in turn claiming priority back to a foreign application filed on December 27, 2000. For purposes of this Declaration, I have assumed December

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

27, 2000 as the effective filing date for the '961 patent. I have cited to the following documents in my analysis below:

<b>Ex. No.</b>	<b>Description of Document</b>
<b>1001</b>	U.S. Patent No. 7,372,961 B2 to Scott A. Vanstone et al. (filed December 26, 2001, issued May 13, 2008)
<b>1003</b>	File Wrapper of U.S. Patent No. 7,372,961
<b>1004</b>	Federal Information Processing (FIPS) Publication 186, <i>Digital Signature Standard (DSS)</i> (May 19, 1994)
<b>1005</b>	Excerpts from Alfred J. Menezes et al., <i>Handbook of Applied Cryptography</i> (1997)
<b>1006</b>	USENET article by Greg Rose to sci.crypt and sci.math, <i>Re: "Card-shuffling" algorithms</i> (March 10, 1993)
<b>1007</b>	Excerpts from Donald E. Knuth, <i>The Art of Computer Programming</i> , Vol. 2 (3d ed. 1998)
<b>1008</b>	Excerpts from Bruce Schneier, <i>Applied Cryptography</i> (2d ed. 1996)
<b>1009</b>	USENET article by Steve Brecher, <i>Re: How can I generate random numbers?</i> (October 12, 1996)
<b>1010</b>	USENET article by Steve Brecher, <i>Re: Help: Random numbers?</i> (November 13, 1996)
<b>1011</b>	USENET article by Trevor L. Jackson, III, <i>Re: Random Generator</i> (April 7, 2000)
<b>1012</b>	Declaration of Aviel Rubin, Ph.D Regarding Claim Construction of the '961 Patent, filed in <i>Blackberry Limited v. Facebook, Inc. et al.</i> , No. 2:18-cv-01844-GW-KS (C.D. Cal.), filed February 28, 2019



Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

<b>Ex. No.</b>	<b>Description of Document</b>
<b>1013</b>	U.S. Patent No. 5,732,138 to Landon Curt Noll et al. (filed January 29, 1996, issued March 24, 1998)
<b>1014</b>	Excerpts from Jenny A. Frstrup, <i>USENET: Netnews for Everyone</i> (1994)
<b>1015</b>	Federal Information Processing (FIPS) Publication 186-2, <i>Digital Signature Standard (DSS)</i> (January 27, 2000) ( <a href="https://web.archive.org/web/20000815231203/http://csrc.nist.gov/fips/fips186-2.pdf">https://web.archive.org/web/20000815231203/http://csrc.nist.gov/fips/fips186-2.pdf</a> )
<b>1016</b>	Brian W. Kernighan & Dennis M. Ritchie, <i>The C Programming Language</i> (2d ed. 1988)
<b>1017</b>	John von Neumann, <i>Various Techniques Used in Connection with Random Digits</i> , Summary by G.E. Forsythe, National Bureau of Standards Applied Math Series, 12 (1951), pp 36-38, reprinted in von Neumann's Collected Works, 5 (1963), Pergamon Press pp 768-770.
<b>1018</b>	Excerpts from Thammavarapu R. N. Rao, <i>Error Coding for Arithmetic Processors</i> (1974)
<b>1019</b>	Excerpts from Nancy A. Floyd, <i>Essentials of Data Processing</i> (1987)
<b>1020</b>	Amendments to Claims and Remarks, October 30, 2007, filed in U.S. Appl. Ser. No. 10/025,924 (issuing as the '961 patent)
<b>1028</b>	Excerpts from <i>Merriam Webster's Collegiate Dictionary</i> (10th ed. 1998)
<b>1029</b>	Excerpts from <i>The American Heritage Dictionary of the English Language</i> (4th ed. 2000)

## **II. PERSON HAVING ORDINARY SKILL IN THE ART**

10. I understand that an assessment of the claims of the '961 patent should be undertaken from the perspective of what would have been known or understood by a person of ordinary skill in the art as of the earliest claimed priority date of the patent claims which, as noted, I have assumed is December 27, 2000.

11. Counsel has advised me that to determine the appropriate level of one of ordinary skill in the art, I may consider the following factors: (a) the types of problems encountered by those working in the field and prior art solutions thereto; (b) the sophistication of the technology in question, and the rapidity with which innovations occur in the field; (c) the educational level of active workers in the field; and (d) the educational level of the inventor.

12. The relevant technical field for the '961 patent is key generation and cryptography. ('961, 1:5-6.) Based on the factors listed above, in my opinion a person of ordinary skill in the art as of December 2000 would have had a bachelor's degree or the equivalent in electrical engineering or computer science (or equivalent degree) and at least two years of experience with computer-based cryptographic techniques. A person could also have qualified as a person of ordinary skill in the art with (1) less technical experience and more formal education (such as a master's

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

degree or Ph.D.) or (2) more technical experience and less formal education. Additionally, the requisite technical experience and formal education could have been acquired concurrently. As part of such person's basic education and/or experience, a person of ordinary skill in the art would have obtained a working knowledge of computer programming and would have learned how to implement cryptographic algorithms in software, how to test those algorithms, and how to adapt them for practical applications (*e.g.*, for digital communications).

13. My opinions regarding the level of ordinary skill in the art are based on, among other things, my more than 20 years of experience in cryptography and related fields, my understanding of the basic qualifications that would be relevant to an engineer or scientist tasked with investigating methods and systems in the relevant area, and my familiarity with the backgrounds of colleagues, co-workers, and employees, both past and present, who work in that area.

14. Although my qualifications and experience exceed those of the hypothetical person having ordinary skill in the art defined above, my analysis and opinions regarding the '961 patent have been based on the perspective of a person of ordinary skill in the art in December 2000.

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

15. My opinions herein regarding the understanding of a person of ordinary skill in the art and my other opinions set forth herein would remain the same if the person of ordinary skill in the art were determined to have somewhat more or less education and/or experience than I have identified above.

16. I have reviewed the “Declaration of Aviel Rubin, Ph.D Regarding Claim Construction of the ’961 Patent,” which contains a declaration by an expert retained by the Patent Owner. (Rubin Decl., Ex. 1012.) Dr. Rubin opines in that declaration that a person of ordinary skill in the art for purposes of the ’961 patent “would have had a bachelor of science degree in Computer Science or the equivalent, and approximately three years of work or research experience in the field of cryptography or an equivalent subject matter; or an advanced degree (masters or doctorate) in Computer Science or the equivalent, and less work or research experience (dependent in part on the level of degrees achieved) in the field of cryptography or an equivalent subject matter.” (Rubin Decl., Ex. 1012, ¶34.)

17. In my opinion, Dr. Rubin’s formulation of a person of ordinary skill in the art is not materially different from mine. His formulation adds one additional year of work or research experience (“approximately three years”), a difference that I do not consider material to my analysis. My opinions as expressed in this

Declaration would therefore remain the same regardless of whether I applied my own formulation or the one provided by Dr. Rubin.

### **III. LEGAL PRINCIPLES USED IN THE ANALYSIS**

18. I am not a patent attorney, nor have I independently researched the law on patent validity. Attorneys for the Petitioner explained certain legal principles to me that I have relied upon in forming my opinions set forth in this report.

#### **B. Prior Art**

19. I understand that the law provides categories of information that constitute prior art that may be used to anticipate or render obvious patent claims. To be prior art to a particular patent, a reference must have been made, known, used, published, or patented, or be the subject of a patent application by another, before the priority date of the patent. I am informed that admissions by the patent application regarding the presence of features in the prior art, whether contained in the patent specification or made during the prosecution history, can be relied upon to show anticipation or obviousness of a claim. I also understand that a person of ordinary skill in the art is presumed to have knowledge of the relevant prior art.

20. I understand that the Petitioner has assumed, for purposes of my Declaration, that the challenged claims of the '961 patent are entitled to a December 27, 2000 priority date.

**C. Claim Construction**

21. I understand that under the applicable legal principles, claim terms are generally given their ordinary and customary meaning, which is the meaning that the term would have to a person of ordinary skill in the art at the time of the invention, *i.e.*, as of the effective filing date of the patent application. I further understand that a person of ordinary skill in the art is deemed to read claim terms not only in the context of the particular claim in which a claim term appears, but in the context of the entire patent, including the specification.

22. I am informed by counsel that the patent specification, under the applicable legal principles, is often the single best guide to the meaning of a claim term, and is thus highly relevant to the interpretation of claim terms. And I understand for terms that do not have a customary meaning within the art, the specification usually supplies the best context of understanding their meaning.

23. I am further informed by counsel that other claims of the patent in question, both asserted and unasserted, can be valuable sources of information as to

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

the meaning of a claim term. Because the claim terms are normally used consistently throughout the patent, the usage of a term in one claim can often illuminate the meaning of the same term in other claims. Differences among claims can also be a useful guide in understanding the meaning of particular claim terms.

24. I understand that the prosecution history can further inform the meaning of the claim language by demonstrating how the inventors understood the invention and whether the inventors limited the invention in the course of prosecution, making the claim scope narrower than it otherwise would be. Extrinsic evidence, such as my expert testimony, may also be consulted in construing the claim terms.

25. I have been informed by counsel that, in *Inter Partes* Review (IPR) proceedings, a claim of a patent shall be construed using the same claim construction standard that would be used to construe the claim in a civil action filed in a U.S. district court (which I understand is called the “*Phillips*” claim construction standard), including construing the claim in accordance with the ordinary and customary meaning of such claim as understood by one of ordinary skill in the art and the prosecution history pertaining to the patent.

26. I have been instructed by counsel to apply the “*Phillips*” claim construction standard for purposes of interpreting the claims in this proceeding, to

the extent they require an explicit construction. The description of the legal principles set forth above provides my understanding of the “Phillips” standard as provided to me by counsel.

**D. Legal Standards for Obviousness**

27. I have been provided the instructions reproduced in part below, taken from the Federal Circuit Bar Association Model Instructions regarding obviousness. I follow these instructions in my analysis, with the caveat that I have been informed that the Patent Office will find a patent claim invalid in *inter partes* review if it concludes that it is more likely than not that the claim is invalid (*i.e.*, a preponderance of the evidence standard), which is a lower burden of proof than the “clear and convincing” standard that is applied in United States district court (and described in the instructions below):

**4.3c OBVIOUSNESS**

Even though an invention may not have been identically disclosed or described before it was made by an inventor, in order to be patentable, the invention must also not have been obvious to a person of ordinary skill in the field of technology of the patent at the time the invention was made.



Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

[Alleged infringer] may establish that a patent claim is invalid by showing, by clear and convincing evidence, that the claimed invention would have been obvious to persons having ordinary skill in the art at the time the invention was made in the field of [insert the field of the invention].

In determining whether a claimed invention is obvious, you must consider the level of ordinary skill in the field [of the invention] that someone would have had at the time the [invention was made] or [patent was filed], the scope and content of the prior art, and any differences between the prior art and the claimed invention.

Keep in mind that the existence of each and every element of the claimed invention in the prior art does not necessarily prove obviousness. Most, if not all, inventions rely on building blocks of prior art. In considering whether a claimed invention is obvious, you may but are not required to find obviousness if you find that at the time of the claimed invention [or the patent's filing date] there was a reason that would have prompted a person having ordinary skill in the field of [the invention] to combine the known elements in a way the claimed invention does, taking into account such factors as (1) whether the claimed invention was merely the predictable result of using prior

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

art elements according to their known function(s); (2) whether the claimed invention provides an obvious solution to a known problem in the relevant field; (3) whether the prior art teaches or suggests the desirability of combining elements claimed in the invention; (4) whether the prior art teaches away from combining elements in the claimed invention; (5) whether it would have been obvious to try the combinations of elements, such as when there is a design need or market pressure to solve a problem and there are a finite number of identified, predictable solutions; and (6) whether the change resulted more from design incentives or other market forces. To find it rendered the invention obvious, you must find that the prior art provided a reasonable expectation of success. Obvious to try is not sufficient in unpredictable technologies.

In determining whether the claimed invention was obvious, consider each claim separately. Do not use hindsight, i.e., consider only what was known at the time of the invention [or the patent's filing date].

In making these assessments, you should take into account any objective evidence (sometimes called “secondary

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

considerations”) that may shed light on the obviousness or not of the claimed invention, such as:

- (a) Whether the invention was commercially successful as a result of the merits of the claimed invention (rather than the result of design needs or market-pressure advertising or similar activities);
- (b) Whether the invention satisfied a long-felt need;
- (c) Whether others had tried and failed to make the invention;
- (d) Whether others invented the invention at roughly the same time;
- (e) Whether others copied the invention;
- (f) Whether there were changes or related technologies or market needs contemporaneous with the invention;
- (g) Whether the invention achieved unexpected results;
- (h) Whether others in the field praised the invention;
- (i) Whether persons having ordinary skill in the art of the invention expressed surprise or disbelief regarding the invention;
- (j) Whether others sought or obtained rights to the patent from the patent holder; and

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

(k) Whether the inventor proceeded contrary to accepted wisdom in the field.

Federal Circuit Bar Association Model Jury Instructions §4.3c (2014).

28. I am also informed that the United States Patent Office supplies its examining corps with a Manual of Patent Examining Procedure that provides exemplary rationales that may support a conclusion of obviousness, including:

(a) Combining prior art elements according to known methods to yield predictable results;

(b) Simple substitution of one known element for another to obtain predictable results;

(c) Use of known technique to improve similar devices (methods, or products) in the same way;

(d) Applying a known technique to a known device (method, or product) ready for improvement to yield predictable results;

(e) “Obvious to try” – choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success;

(f) Known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if

the variations are predictable to one of ordinary skill in the art; or

(g) Some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

MPEP §2143. I apply these principles in my analysis below.

#### **IV. RELEVANT TECHNOLOGY BACKGROUND**

29. The '961 patent, entitled "Method of Public Key Generation," purports to disclose and claim a method for generation of a key for use in a cryptographic function. ('961, claim 1.) The '961 patent focuses primarily on a single aspect of cryptographic functions – the generation of an appropriate cryptographic key. The patent thus falls generally in the fields of key generation and cryptography.

30. Cryptographic keys are often generated using a random number generator. ('961, 1:41-43) Random number generation has long been practiced in computer science and cryptography. In this section, I will provide a brief background discussion of technologies pertinent to the '961 patent prior to December 2000.

## **B. Random Number Generation**

31. Random number generation refers to the generation of a sequence of numbers that are or appear random, *i.e.*, lack any pattern. Random number generation was a highly developed field within mathematics and computer science prior to the filing of the '961 patent.

32. A simple example of a random number generator is a die with six sides showing 0, 1, 2, 3, 4, and 5.<sup>1</sup> Each roll of the die generates a random number in the range from 0 to 5. If the die is unweighted, then each side is equally likely, *i.e.*, each roll is equally likely to result in any of the numbers 0, 1, 2, 3, 4, or 5. When each value in a given set of possible values is equally likely to occur, the output of the random number generator



---

<sup>1</sup> In my discussion, I will be using a die with faces showing 0-5 instead of 1-6 because the lowest unsigned value representable in a computer is zero, not one. Thus, unsigned numbers on computers range from 0 to their maximum value.

is said to have a “uniform distribution” in that set. When I refer to this six-sided die further below, I always assume that it is an unweighted die for which each number is equally likely, and thus the outcome is uniformly distributed in the range of 0-5.

**C. Cryptography and Its Reliance on Random Number Generators**

33. Computer-based techniques have long existed for generating and using random numbers. Random numbers are used for a number of computer-based applications, most notably here, cryptography.<sup>2</sup>

34. The term “cryptography” is often associated with *encryption*, the process of converting original information (known as “plaintext”) into a scrambled

---

<sup>2</sup> Cryptography is a broad field in computer science, both in academia and in many parts of the computer hardware and software industry. I note that the claims of the ’961 patent do not recite performing cryptographic functions on input data – they focus exclusively on using random numbers to generate cryptographic keys. The claims do not recite how those keys, once generated, are subsequently used. Accordingly, I will only provide a brief and high-level overview about cryptography.

or otherwise unintelligible form (known as “ciphertext”) to protect against discovery of the plaintext by an unauthorized third party. (See Bruce Schneier, *Applied Cryptography* (2d ed. 1996), Ex. 1008, pp.1-2.) Encryption is commonly used to send private or secure messages over public networks such as the Internet. When the contents of a message have been *encrypted*, they will appear unintelligible to someone who may intercept the resulting encrypted message on its way to the recipient. But once the encrypted message reaches its intended destination, the recipient can *decrypt* it by using a process that turns the unintelligible data back into the original message. Encryption thus provides a way to protect the privacy of messages even when sent over an insecure communications channel.

35. Encryption and decryption techniques have existed for hundreds of years. Long before the introduction of computers, people used “substitution ciphers” to replace each character in an original message with a substitute character. (Schneier, Ex. 1008, pp.10-11.) These techniques generally required that the sender and a recipient possess a shared “codebook” for use in performing the substitution (by the sender), and reversing the substitution (by the recipient). More recently, the Germans used mechanical encryption technologies to encrypt military communications during World War II, most famously with the Enigma machine.



(Schneier, p.13.) Encryption algorithms today are primarily implemented using computers, which are well-suited to applying algorithms to analyze and transform data.

36. Encryption prevents an unauthorized party from learning the original content of a message, but it may not protect against *tampering* with a message before it reaches the recipient. For example, if a malicious third party intercepts a message, she could alter it or even replace the contents of the message with some other message. Another cryptographic technique, known as *digital signatures*, is designed to address these types of problems.

37. Generally speaking, a digital signature is a piece of information that, using a cryptographic algorithm, can be used to verify the identity of the sender and the integrity of the data. In simpler terms, a recipient can use a digital signature to verify that a message actually came from the purported sender, and that the message content was not altered in transit.

38. Digital signature algorithms typically rely on various “**keys**,” which are represented as a string of bits (ones and zeroes) and can be viewed as a (possibly very large) non-negative integer. For example, if a key is 160 bits long, then it has  $2^{160}$  possible values, which can be viewed as being in the range from zero all the way

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

to  $2^{160}$  minus 1.<sup>3</sup>

39. But in order for a cryptographic technique to be secure, the key needs to be generated in such a way that it is nearly impossible for a third party to guess or ascertain. It goes without saying that the more closely a key follows some predictable or ascertainable pattern, the greater the likelihood that an unauthorized third party can recover the key. Accordingly, many cryptographic systems rely

---

<sup>3</sup> As the '961 patent explains, many cryptographic systems utilize a “public key” and a “private key.” A public key, as its name implies, is a cryptographic key that someone makes available to others. For example, a recipient can provide its “public key” to a potential sender, who can use the public key to encrypt data for transmission to the recipient. The recipient, in turn, uses the “private key” (known only to her) to decrypt the data. ('961, 1:10-16, 1:26-31.) Public key cryptosystems date back to at least the 1970s. Because the challenged claims do not recite any particular type of cryptographic keys, or specify particular use of a key after its generation, public key cryptography need not be extensively discussed.

heavily on *random number generators* to generate cryptographic keys that, to the extent feasible, are not predictable and that follow no discernable pattern. In fact, many cryptographic systems require keys that are, or at least to appear to be, uniformly distributed in some range.

40. It is difficult or impossible to generate *truly* random numbers using a computer alone. As one well-known cryptographer once remarked, “it is impossible to produce something truly random on a computer.” (Schneier, Ex. 1008, p.44, §2.8.) This is because “[c]omputers are deterministic beasts: Stuff goes in on one end, completely predictable operations occur inside, and different stuff comes out the other end. . . . Put the same stuff into two identical computers, and the same stuff comes out of both of them.” (*Id.*) “And if something is predictable, it can’t be random.” (*Id.*) As John von Neumann famously wrote in 1951, “[a]ny one who considers arithmetical methods of producing random digits is, of course, in a state of sin.” (John von Neumann, *Various Techniques Used in Connection with Random Digits* (1951), Ex. 1017, p.768.)

41. Computer programs therefore often rely on some source of external

randomness (or entropy) to use as the basis for random number generation.<sup>4</sup> To take just a few examples, it was well-known for computers to rely on user input (such as movement of the mouse or the user's typing speed), timing information from the computer system (such as the current setting of the system clock or network interrupts), or additional hardware devices that detect "noisy" (and statistically random) signals and translate them into random sequences of bits. (*See* Schneier, pp.423-425.) Any suitable external source of randomness could be employed.<sup>5</sup> Note

---

<sup>4</sup> Techniques for generating random numbers were well-known to persons of ordinary skill in the art. The '961 patent does not purport to have made any advancements in this area; the claims merely recite a "random number generator" with no further specification. The specification shows random number generator **26** as a nondescript black box in Figure 1. ('961, Figure 1.)

<sup>5</sup> For example, the random number system in U.S. Patent No. 5,732,138 (Ex. 1013) can use digitized pictures of a moving freeway, clouds, or lava lamps to provide an external source of entropy. (Ex. 1013, 3:6-13.)

that such external sources of randomness are not necessarily uniformly distributed. If a uniform distribution is required, the random number generator is designed to process the randomness in some way so as to yield a uniform distribution or a suitable approximation thereof.

**D. Modular Reduction and the Modulo Bias Problem**


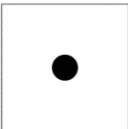
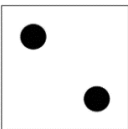
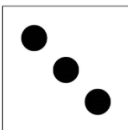
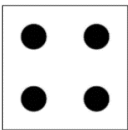

42. Random number generators typically output numbers that are uniformly distributed within a large, specified range. For example, if a random number generator generates 32-bit numbers, then it can generate values ranging from zero to  $2^{32}$  minus 1, which is 4,294,967,295.

43. But many applications need random numbers that fall within a much smaller range than the range of the underlying random number generator. To use a very simple example, suppose someone wants to generate random numbers from 0 to 2 by throwing the six-sided die mentioned above, having sides showing 0-5. Half of the sides yield values from 0 to 2, within the desired range. But the other half of the sides result in values from 3 to 5, which fall *outside* the desired range.

44. One common and well-known solution to this problem is to perform a well-known mathematical operation known as a *modular reduction*. A modular reduction computes the remainder when one number is divided by another. The

concept behind a modular reduction would have been familiar to elementary schoolchildren who have learned long division: For example, when dividing 5 by 3, the quotient is 1 *with a remainder of 2*. Modular reduction returns this remainder. This operation is often expressed using the mathematical notation “**mod**,” so the example earlier would be expressed as “**5 mod 3 = 2**.” (Schneier, Ex. 1008, p.242, §11.3.) The divisor (3 in the example) is called the *modulus*. The operation “a mod n,” and the “mod n operation” applied to a, mean computing the remainder when a is divided by n. (*Id.*) I also refer to this (modular reduction) as a modulo operation, and thus a “modulo n” operation means modular reduction with a modulus of n.


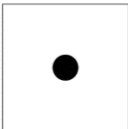
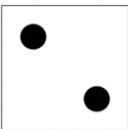
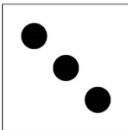
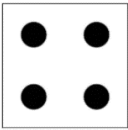
45. A modular reduction is often used with random number generators to generate numbers that fall within a desired range. This is because the result of a modular reduction will *always* be less than the modulus. For example, the operation “**x mod 3**” can only yield three possible values – 0, 1, or 2 – regardless of the value of “x.” Returning to the example of generating a random number from 0 to 2 using the six-sided die with faces from 0-5, one could simply apply reduction modulo 3 to the outcome of rolling the die, and the result would fall within the desired range of 0 to 2. This is shown in the table below, which shows how this modular operation results in a sequence of “repeating” values in the range of 0 to 2.

Roll of the Die	Modulo Operation (Die Value Mod 3)	Resulting Value
	$0 \bmod 3 = 0$	0
	$1 \bmod 3 = 1$	1
	$2 \bmod 3 = 2$	2
	$3 \bmod 3 = 0$	0
	$4 \bmod 3 = 1$	1
	$5 \bmod 3 = 2$	2

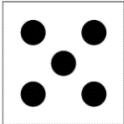
46. Here, the operation “**x mod 3**” transforms values outside the desired range – namely, 3, 4, and 5 – into 0, 1, and 2, respectively. The operation created a *uniform distribution* of random numbers here because each roll of the die uniformly results in a number in the range of 0 to 5, and thus has an equal or “uniform”

likelihood of yielding a 0, 1, or 2 after the modular reduction.

47. But suppose I modified the example slightly – instead of using the six-sided die to generate a number from 0 to 2, I use it to generate a number from 0 to 3 by reducing modulo 4. This situation results in *modulo bias* because 6 is not evenly divisible by 4. As the table below shows, performing the modular reduction in this situation does not result in a uniform distribution of numbers in the range of 0 to 3:

Roll of the Die	Modulo Operation (Die Value Mod 4)	Resulting Value
	$0 \bmod 4 = 0$	0
	$1 \bmod 4 = 1$	1
	$2 \bmod 4 = 2$	2
	$3 \bmod 4 = 3$	3
	$4 \bmod 4 = 0$	0



Roll of the Die	Modulo Operation (Die Value Mod 4)	Resulting Value
	$5 \bmod 4 = 1$	1

48. The operation shown above does not produce random numbers uniformly distributed in the range from 0-3 because *two* positions on the die (0 and 4) yield 0, *two* positions (1 and 5) yield 1, but only *one* position yields 2 and only *one* yields 3. This candidate random number generator thus exhibits **modulo bias** because it is *twice* as likely to output a 0 or a 1 than it is to output a 2 or a 3.

49. This simplified example illustrates a clear modulo bias because the desired range and the maximum underlying random number were set to small values. In real-world random number generators used by cryptographic systems, the desired random number range and the range of the underlying random number generator would be very large numbers. If the size of the underlying random number generator's range is not evenly divisible by the size of the desired range, modulo bias could occur if modular operations are applied as described above, but the bias may not be as detectable as in my simplified example.

**E. Rejecting Random Numbers Above a Desired Range, aka Rejection Sampling**

50. Modular arithmetic is not the only way to generate a random number that falls within a desired range. Another known technique is to simply throw out, or reject, any number output by the underlying random number generator that lies outside the desired range, and to keep generating new random numbers until one is obtained that falls within the desired range. More specifically, one could (1) generate a random number, (2) check whether the number falls within the desired range, (3) if not, discard the random number and go back to (1), or (4) accept and use the number. Using our example above of using a six-sided die to generate a random number from 0 to 3, we would roll the die until it produced a number from 0 to 3, rejecting any rolls of a 4 or a 5. This technique avoids the modulo bias problem discussed above, because it does not rely on modular reduction to obtain values within the desired range. If the underlying random number generator produces a uniform distribution of random numbers within its range, then the process just described will output a uniform distribution of random numbers within the desired range, because the outputs outside the desired range are simply discarded and not used.

51. The idea of rejecting random values that fall outside a desired range was a well-known concept in computer science that dates back to at least the early 1950s. In a famous 1951 article entitled *Various Techniques Used in Connection with Random Digits*, mathematician John von Neumann proposed a technique commonly referred to as **rejection sampling** for generating a uniform distribution. He provides a simple example of using a biased coin to generate unbiased random data:

To cite a human example, for simplicity, in tossing a coin it is probably easier to make two consecutive tosses independent than to toss heads with a probability exactly one-half. If independence of successive tosses is assumed, we can construct a 50-50 chance out of even a badly biased coin by tossing twice. If we get heads-heads or tails-tails, we reject the tosses and try again. If we get heads-tails (or tails-heads), we accept the result as heads (or tails). The resulting process is rigorously unbiased...

(John von Neumann, *Various Techniques Used in Connection with Random Digits* (1951), Ex. 1017, at p.768 (underlining added).) This method produces a uniform distribution over the set of outcomes “heads-tails” and “tails-heads” by rejecting the outcomes “heads-heads” and “tails-tails” which are not in that set. One can thus use this technique to generate a sequence of uniformly distributed bits by associating

heads-tails with “0” and tails-heads with “1.” This simple example shows how rejecting certain results and accepting others can be used to create a uniform distribution of random values in a desired set by sampling from a distribution of random values in a larger set. By accepting only outcomes that occur with equal probability (*i.e.*, sequences of heads-tails or tails-heads), one can generate uniformly distributed results even from a biased coin.<sup>6</sup>

52. In that same 1951 article, von Neumann expanded this technique to describe a way to generate random numbers following an arbitrary desired distribution  $f$  (including as a special case the uniform distribution in some desired range) given the ability to generate random numbers uniformly in the real interval  $(0, 1)$ . The idea is to first choose a “scaling factor”  $a$  such that  $a f(x) \leq 1$  for all  $x$ . Then generate  $X$  and  $Y$ , each with the uniform distribution in the real interval  $(0, 1)$ .

---

<sup>6</sup> Of course, the efficiency of this technique will depend on how biased the coin is. The more biased the coin, the greater the number of tosses would be needed to obtain the desired amount of random data.

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

If  $Y > af(X)$ , then reject and start the process again; otherwise, accept  $X$  as the result. (*Id.*, p.769 (“One trick is to pick a scaling factor  $a$  such that  $af(x) \leq 1$ , and then to produce two random numbers,  $X$  and  $Y$ , from a uniform distribution on  $(0, 1)$ . If  $Y > af(X)$ , we reject the pair and call for a new pair. If  $Y \leq af(X)$ , we accept  $X$ .”) (italics in original).)

53. The famous book by Donald Knuth, *The Art of Computer Programming*, Volume 2 (3d ed. 1998) refers to von Neumann’s technique as the *rejection method* and summarizes it as a technique for “obtaining a complicated density from another one that ‘encloses’ it.” (Donald E. Knuth, *The Art of Computer Programming*, Vol. 2 (3d ed. 1998), Ex. 1007, p.125.) As “a simple example of such an approach,” Knuth explains that one can generate a point uniformly distributed in a circle that sits within a larger square by “first generating a random point in a larger square, rejecting it and starting over again if the point was outside the circle.” (*Id.*) Knuth also describes a generalization of von Neumann’s method that allows for generation of random values following an arbitrary distribution  $f$  given the ability to generate random values following some arbitrary distribution  $g$ . (*Id.*)

54. The rejection method described by von Neumann, and the generalization thereof described by Knuth, were actually far more complex than the

simplistic technique claimed decades later in the '961 patent. The '961 patent employs perhaps the simplest rejection-type technique imaginable – checking whether a generated random number lies in a desired range (which in the patent is determined by whether it is less than the integer  $q$ ), accepting the generated number if it does, and otherwise rejecting it and generating another random number.<sup>7</sup> ('961, Fig 2, claim 1 (“determining whether said output  $H(SV)$  is less than said order  $q$  prior to reducing mod  $q$ ; accepting said output  $H(SV)$  for use as said key  $k$  if the value of said output  $H(SV)$  is less than said order  $q$ ; rejecting said output  $H(SV)$  as said key if said value is not less than said order  $q$ ”).) Nevertheless, even the simplistic rejection technique described in the '961 patent was known in the art.

55. The technique used in the '961 patent was described in the *Handbook*

---

<sup>7</sup> The '961 patent does not purport to describe a novel underlying random number generator; rather, the patent takes for granted the existence of a suitable random number generator. The '961 patent also does not employ techniques for minimizing the likelihood of rejecting the output of this random number generator.

of *Applied Cryptography* (1997), whose authors include two of the '961 patent's named inventors, which explains that "[a] *random bit generator* is a device or algorithm which outputs a sequence of statistically independent and unbiased binary digits." (Alfred J. Menezes et al., *Handbook of Applied Cryptography* (1997), Ex. 1005, p.170, §5.1 (italics in original).) Menezes explains that a random bit generator can be used to generate uniformly distributed random numbers in the desired range of 0 to  $n$  by rejecting any values that exceed  $n$ :

A random bit generator can be used to generate (uniformly distributed) random numbers. For example, a random integer in the interval  $[0, n]$  can be obtained by generating a random bit sequence of length  $\lfloor \lg n \rfloor + 1$ , and converting it to an integer; if the resulting integer exceeds  $n$ , one option is to discard it and generate a new random bit sequence.

(Menezes, Ex. 1005, p.170, §5.2 (underlining added).) In other words, one can generate uniformly distributed random numbers within a desired range by repeatedly generating random numbers that are uniformly distributed in some larger range, and accepting only those values that fall within the desired range. This exceedingly simple technique is what was later claimed in the '961 patent.

56. At least three programmers on the Internet, apparently working independently, came up with and published this same technique. First, on March

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

10, 1993, Greg Rose posted comments to the `sci.crypt` and `sci.math` newsgroups on USENET, in response to an article by another programmer, about the following line of code (written in the C programming language):

```
r = lrand48() % 3;
```

(USENET article by Greg Rose to `sci.crypt` and `sci.math`, *Re: "Card-shuffling" algorithms* (March 10, 1993), Ex. 1006, p.1). This code generates random numbers from 0 to 2 via a commonly-used random number generator (`lrnd48()`) available as part of the C standard code libraries. In English, the code takes the random number output by `lrnd48()`, and performs a modular reduction on it (with modulus 3) to obtain a result ranging from 0 to 2. Rose explains that this approach created a modulo bias problem, and he suggested the solution:

Technically, there IS a problem with the above generator... When the desired interval doesn't exactly divide the interval you already have, the smaller numbers are more likely than the larger ones. What you really have to do is throw away any result from the original generator that is too big. The worst case of this, by one estimate, is when you are trying to generate random numbers in an interval roughly 2/3 of the input interval. In this case, half of the numbers are twice as likely as the other half. What is worse is that merely looking at the resulting numbers (with an eyeball I mean) probably won't make this apparent.



Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

(Rose, Ex. 1006, p.2 (underlining added).) Rose went on to disclose exemplary C source code that generates a random number and throws it away if it is “too big,” *i.e.*, if it exceeds a desired maximum:

```
/*
 * Roll a random number [0..n-1].
 * Avoid the slight bias to low numbers.
 */
int
roll(int n)
{
    long rval;
    long biggest;
#define BIG 0x7FFFFFFFL /* biggest value returned by random() */
    if (n == 0)
        return 0;
    biggest = (BIG / n) * n;
    do {
        rval = random();
    } while (rval >= biggest);
    return rval % n;
}
```

(*Id.*) I will discuss this code in detail below. In broad overview, it defines a function “**roll**” that attempts to generate a uniformly distributed random number in the range of 0 to “**n**” minus one. In the function, the constant “**BIG**” equals the largest value that can be generated by the underlying random number generator `random()` (which is defined in Rose’s function as 2,147,483,647). The variable “**biggest**” is assigned a value representing the largest multiple of “**n**” that is less than or equal to “**BIG**.” The variable “**biggest**” thus defines a desired range of random values from 0 to “**biggest**” minus one. The code then repeatedly generates random numbers using

the underlying random number generator `random()` until it obtains one less than “**biggest**.” Once a number less than `biggest` is generated by the underlying random number generator, the number is reduced modulo `n` to produce the actual output number (a number between 0 and `n-1`). Because output values greater than or equal to “**biggest**” are rejected – values that could otherwise have introduced modulo bias – each number between 0 and “`n`” minus one has an equal probability of being returned by the function “**roll**.”

57. This code can easily be applied to my earlier example of obtaining a random number that is uniformly distributed in the range from 0 to 3 using a six-sided die with values 0-5. Under that scenario, the constant “BIG” would be assigned 5 (the maximum number generated by the die), “`n`” would be 4 (specifying our desired range), and thus, “**biggest**” is 4 – the largest multiple of 4 that is less than or equal to 5. Rose’s “**roll**” function would then call `random()` to generate a random number in the range 0-5 until it generates a number less than 4 (**biggest**). Thus, if it did not generate a 0, 1, 2, or 3, “roll” would reject the number and try again by generating another random number.

58. Greg Rose was not the only programmer on the Internet to recognize the modulo bias problem and to suggest this solution. In 1996, a programmer named

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

Steve Brecher posted an article providing substantially the same solution in response to a line of code that another programmer had posted:

```
rand( ) %129
```

(USENET article by Steve Brecher, *Re: How can I generate random numbers?* (October 12, 1996), Ex. 1009, p.1; *see also* Steve Brecher, *Re: Help: Random numbers* (November 13, 1996), Ex. 1010 (another similar post by Steve Brecher).)

This line of code uses the standard C random number generator (rand()) to output a random number in the range of 0 to 128. Similar to the code in Rose, this code takes the output of the random number generator (rand()) and performs a reduction modulo 129 (“%129”) on it to ensure a range from 0 to 128. Brecher explained that when the maximum output of rand() is 65,535 – which is not evenly divisible by 129 – the code created a potential for modulo bias:

Consider the case in which RAND\_MAX is 65535. The largest multiple of 129 in 65535 is 65532 (508 times 129). So if you enumerate the possible return values of rand() and map each to the result of rand()%129, it would look like:

```
0 -> 0
1 -> 1
2 -> 2
...
128 -> 128
```

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

```
129 -> 0
...
65532 -> 0
65533 -> 1
65534 -> 2
65535 -> 3
```

and hence if you tabulate [sic] the distribution of results of `rand()%129`, the values 1, 2, and 3 will occur 509 times each, while the values 0 and 4 through 128 will occur 508 times each.<sup>8</sup>

(Brecher, Ex. 1009, p.1) Brecher went on to provide a function written in C++, very similar in design to the one Rose published three years earlier, for generating random numbers and “reject[ing] values of the underlying generator which are larger than or equal to the largest multiple of the ceiling.” (*Id.*)

59. And on April 7, 2000, more than three years after Brecher, Trevor

---

<sup>8</sup> I note that Brecher’s calculation here is slightly off, since in fact the value 0 will also occur 509 times. But the larger point (namely, the fact that there is a modulo bias) is unchanged.

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

Jackson III posted an article on USENET identifying the same problem and substantially the same solution as Rose and Brecher. Jackson's USENET article explained that "[a] way to eliminate this bias is to discard RNG [i.e., random number generator] outputs larger than the greatest range multiple smaller than RAND\_MAX."<sup>9</sup> (USENET article by Trevor L. Jackson, III, *Re: Random Generator* (April 7, 2000), Ex. 1011, pp.1-2.) Jackson also provided source code similar to the code posted by Brecher and Rose for implementing this technique. (*Id.*, p.2.)

60. I note that the Rose, Brecher, and Jackson articles make no mention of each other or their respective articles, which were each published years apart and on different USENET newsgroups. It therefore appears that these three programmers independently identified and described substantially the same solution to the modulo

---

<sup>9</sup> The associated code discards values that are larger than or equal to the greatest range multiple smaller than or equal to RAND\_MAX, which achieves the desired result.

bias problem for random number generation. A person of ordinary skill in the art would not have found this surprising, considering that modular reduction was well-known, and the technique that Rose, Brecher, and Jackson employ was a well-known (and highly simplified) variation of the decades-old rejection method. And each article shows that the rejection technique could be implemented with only a few lines of code.

## V. OVERVIEW OF THE '961 PATENT

### A. The Specification

61. The '961 patent, entitled "Method of Public Key Generation," generally purports to describe a method for avoiding potential bias in the generation of a cryptographic key. More specifically, the patent purports to remedy modulo bias in the generation of a key  $k$  in connection with the Digital Signature Algorithm (DSA), described in a well-known standard known as the **Digital Signature Standard (DSS)**. DSS was published by the National Institute of Standards and Technology (NIST), which is part of the United States Department of Commerce. (Federal Information Processing (FIPS) Publication 186, *Digital Signature Standard (DSS)* (May 19, 1994), Ex. 1004, p.1.) The signature algorithm is commonly known as the DSA, and the published document that describes the standard is known as the DSS.

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

The DSS explains that “[a] digital signature is an electronic analogue of a written signature in that the digital signature can be used in proving to the recipient or a third party that the message was, in fact, signed by the originator.” (DSS, Ex. 1004, p.5, §2.) “Digital signatures may also be generated for stored data and programs so that the integrity of the data and programs may be verified at any later time.” (*Id.*)

62. The DSA and DSS were not inventions of the '961 patent. By the time the earliest application for the '961 patent was filed in December 2000, NIST had already published three versions of DSS – in 1994 (FIPS 186), 1998 (FIPS 186-1), and 2000 (FIPS 186-2) (Ex. 1015). The Background section of the '961 patent specification specifically cites and discusses FIPS 186-2, the 2000 version of DSS. ('961, 2:36-46.) For purposes of my analysis, I will cite primarily to the first version of DSS (FIPS 186/1994), because it is the oldest version of the DSA/DSS. (DSS, Ex. 1004.) The description of how to generate keys in DSS, as summarized in the '961 patent, did not materially change between FIPS 186 (1994) and FIPS 186-2 (2000).

63. The '961 patent focuses on one narrow aspect of DSS – the generation of the ephemeral cryptographic key,  $k$ . As summarized by the '961 patent, the DSS specifies a method for generation of key  $k$  using a random number generator and a

hashing algorithm, and requires that the key  $k$  be less than “ $q$ ”, a large prime number. (’961, 1:56-64, 2:40-43; DSS, Ex. 1004, pp.6, 13-15.) As explained in the ’961 specification:

One such implementation [of the DSA] is the DSS mandated by the National Institute of Standards and Technology (NIST) FIPS 186-2 Standard. The DSS stipulates the manner in which an integer is to be selected for use as a private key. A seed value,  $SV$ , is generated from a random number generator which is then hashed by a SHA-1 hash function to yield a bit string of predetermined length, typically 160 bits. The bit string represents an integer between 0 and  $2^{160}-1$ . However this integer could be greater than the prime  $q$  and so the DSS requires the reduction of the integer mod  $q$ , i.e.  $k=SHA-1(seed) \bmod q$ .

(’961, 2:36-46 (underlining added).) To summarize, the DSS first specifies generating a random number **seed**, and then computing a hash of the random number **SHA-1(seed)**. Because the value of  $SHA-1(seed)$  may be larger than the desired range (as determined by the “prime  $q$ ”), a “**mod  $q$** ” operation is performed to compute the value of the cryptographic key  $k$ , i.e.,  **$k=SHA-1(seed) \bmod q$** . (’961, 2:44-46.) Computing “**SHA-1(seed) mod  $q$** ,” as explained extensively above, ensures that the value of the key  $k$  will fall within the desired range required by the DSA, i.e., from 0 to  $q-1$ .



64. But as explained in detail above, this method for generating  $k$  could introduce some modulo bias in favor of values on the smaller end of the range. The '961 patent acknowledges that the possibility of such bias was known. ('961, 2:53-55 (“With this algorithm it is to be expected that more values will lie in the first interval than the second and therefore there is a potential bias in the selection of  $k$ .”).)

65. The apparent impetus behind the patent was work by a third party, Dr. Daniel Bleichenbacher, who attempted to quantify the bias and suggested that it might reduce the effort needed to recover a private key in DSA. The patent explains that “[r]ecent work by Daniel Bleichenbacher,” a known cryptographer who is not named as an inventor on the '961 patent, “suggests that the modular reduction to obtain  $k$  introduces sufficient bias in to the selection of  $k$  that an examination of  $2^{22}$  signatures could yield the private key  $d$  in  $2^{64}$  steps using  $2^{40}$  memory units.” ('961, 2:56-59.) “This suggests that there is a need for the careful selection of the ephemeral key  $k$ .” ('961, 2:60-61.)

66. The '961 patent purports to solve the problem identified by Bleichenbacher in the same manner as the prior art – instead of relying on a modular reduction to obtain values within the desired range (from 0 to  $q-1$ ), simply *reject* any

random numbers that are not less than  $q$ , and keep generating new random numbers until one is obtained that is less than  $q$ . ('961, 4:11-17.) This technique is reflected in Figure 2:

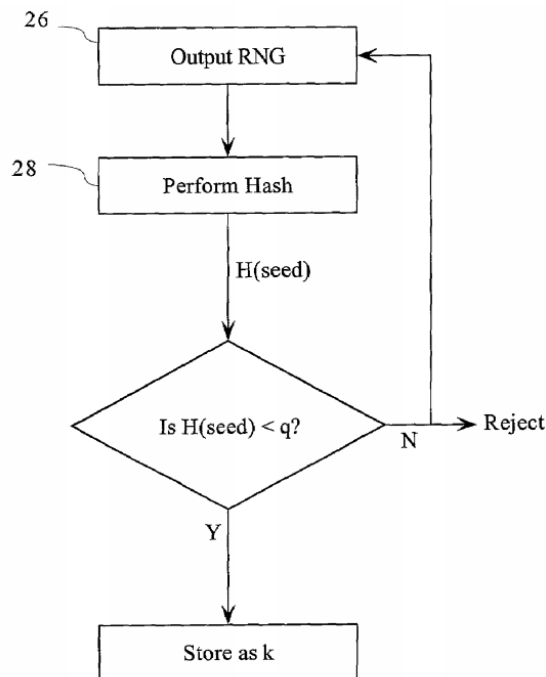


Fig. 2

('961, Fig. 2.) Figure 2 provides “a flow chart showing a first embodiment of key generation” ('961, 3:12-13), and that embodiment is the one reflected in the challenged claims. The method “originates by obtaining a seed value (SV) from the random number generator **26**.” ('961, 3:64-66.) The number output from the random number generator (RNG) is then provided to hash function **28** in order to

produce an output – **H(seed)** –of the correct length. ('961, 4:6-10.) This is the same value represented as “SHA-1(seed)” in the Background description of DSA. ('961, 2:40-46.) The '961 patent explains that the hashing function is used “to yield a bit string of predetermined length, typically 160 bits.” ('961, 2:40-43, 3:38-41.)

67. The remaining steps in Figure 2 perform the simplified rejection technique of the prior art – namely, they check if **H(seed)** is less than **q**, where the maximum desired value is  $q-1$ . If the answer is “yes,” **H(seed)** is accepted and used as the cryptographic key **k**. Otherwise the value **H(seed)** “is rejected and the random number generator is conditioned to generate a new value which is again hashed by the function **28** and tested. This loop continues until a satisfactory value is obtained.” ('961, 4:13-17.)

## **B. The Challenged Claims**

68. My Declaration addresses claims 1, 2, 5, 15, 16, 19, 23, 24, and 27, of which claims 1, 15, and 23 are independent. Claims 1, 2, and 5 are representative and reflect the first embodiment from the specification described above:

1. A method of generating a key **k** for use in a cryptographic function performed over a group of order **q**, said method including the steps of:

[a] generating a seed value **SV** from a random number generator;

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

- [b] performing a hash function  $H()$  on said seed value SV to provide an output  $H(SV)$ ;
  - [c] determining whether said output  $H(SV)$  is less than said order  $q$  prior to reducing mod  $q$ ;
  - [d] accepting said output  $H(SV)$  for use as said key  $k$  if the value of said output  $H(SV)$  is less than said order  $q$ ;
  - [e] rejecting said output  $H(SV)$  as said key if said value is not less than said order  $q$ ;
  - [f] if said output  $H(SV)$  is rejected, repeating said method; and
  - [g] if said output  $H(SV)$  is accepted, providing said key  $k$  for use in performing said cryptographic function, wherein said key  $k$  is equal to said output  $H(SV)$ .
2. The method of claim 1 wherein another seed value is generated by said random number generator if said output is rejected.
5. The method of claim 1 wherein said order  $q$  is a prime number represented by a bit string of predetermined length  $L$ .

(’961, 5:33-54, 5:60-62.) Many of the limitations in these claims recite features that are admitted prior art from the DSA/DSS, including everything in the preamble 1[Pre] of claim 1, limitations 1[a], 1[b], and the extra feature added by dependent claim 5. (’961, 1:52-64, 2:36-44.) The other two groups of challenged claims (*i.e.*,

claims 15/16/19 and claims 23/24/27) recite the computer readable medium and apparatus claims corresponding to the subject matter in claims 1, 2, and 5, respectively.

**C. Interpretation of Claim Limitations in the '961 Patent**

69. I have been asked to provide an opinion regarding the interpretation of one phrase recited in each independent challenged claim of the '961 patent – “**reducing mod  $q$** .” I understand that the construction of this phrase is the subject of an ongoing dispute in the litigation between the Petitioner and the Patent Owner. I have reviewed a document entitled “Declaration of Aviel Rubin, Ph.D Regarding Claim Construction of the '961 Patent,” filed in the district court on February 28, 2019, which I understand expresses the Patent Owner’s positions. (Rubin Decl., Ex. 1012.)

70. Dr. Rubin opines that: “‘reducing mod  $q$ ’ has a plain and ordinary meaning that refers to lowering a numerical value based on a modulo  $q$ .” (Rubin Declaration, Ex. 1012, ¶55.) I respectfully disagree. As I explain below, in the field of cryptography, the words “reducing” and “reduction” when used in conjunction with “mod  $q$ ” do not require lowering of a numerical value based on a modulus  $q$ .

71. A modular reduction, expressed mathematically as “ $a \bmod b$ ,” returns the remainder when  $a$  is divided by  $b$ . It will generate a lower number when  $a \geq b$ , but it

will not generate a lower number when  $a < b$ . A modular reduction therefore may or may not result in a lower value, depending on the inputs to the operation. Nonetheless, persons of ordinary skill in the art refer to “ $a \bmod b$ ” as performing a “modular reduction” or a “reduction modulo  $b$ ” in either circumstance, regardless of whether or not the resulting value is less than the original “ $a$ ” value.

72. As a simple example, assume we are using a modulus of 3 (i.e.,  $b=3$ ). Then if  $a$  is 5, the result of computing “ $5 \bmod 3$ ” is 2, because 5 divided by 3 returns 1, with a remainder of 2. Modular reduction in this case does result in a lower value. However, if  $a$  is 2, then the result of computing “ $2 \bmod 3$ ” is 2, because 2 divided by 3 returns 0, with a remainder of 2. Both are examples of “modular reduction” based on the modulus 3, or just “reducing modulo 3.”

73. Dr. Rubin’s approach to determining the meaning of “reducing mod  $q$ ” is to construe the word “reducing” in isolation using lay-person dictionary definitions. (Rubin Decl., Ex. 1012, ¶¶53-54.) I disagree with this approach. Construing the term “reducing mod  $q$ ” (also called “modular reduction” using a modulus of  $q$ ) requires interpreting the entire term, as the entire term is well-known and defined in the field of cryptography. Thus, treating the parts of the term separately does not capture the true technical meaning of the larger “reducing mod  $q$ ” phrase.

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

74. Dr. Rubin's support for his definition of the word "reducing" in the phrase "reducing mod  $q$ " consists entirely of dictionary definitions for the isolated word "reduce" or "reducing," none of which use the phrase to refer to a modular operation or as it would be understood in the context in which it appears in the claims. (Rubin Decl., Ex. 1012, ¶54.)

75. In the context of mathematics and cryptography, the phrase "reducing mod  $q$ " simply refers to performing the mod  $q$  operation, *i.e.*, computing the remainder when dividing a value by  $q$ . As explained extensively above, the operation expressed as "**a mod b**" *will* generate a smaller number when  $a \geq b$ , but *will not* when  $b > a$ . A modular reduction therefore may or may not result in a reduced value, depending on the values that are input into the operation.

76. The fact that "reducing mod  $q$ " simply refers to computing the remainder upon division by  $q$  is clear from the '961 patent. The specification refers to reducing mod  $q$  only once: "However this integer [SHA-1(seed)] could be greater than the prime  $q$  and so the DSS requires the reduction of the integer mod  $q$ , **i.e.**  $k = \text{SHA-1}(\text{seed}) \bmod q$ ." ('961, 2:44-46 (emphases added).) This sentence treats "reduction of the integer mod  $q$ " as synonymous with computing " $\text{SHA-1}(\text{seed}) \bmod q$ ". Because the integer  $\text{SHA-1}(\text{seed})$  *could* be greater than  $q$ , it must be reduced modulo  $q$ . If  $\text{SHA-1}(\text{seed})$  was

less than  $q$ , reducing it modulo  $q$  does not lower that value, but if  $\text{SHA-1}(\text{seed})$  was greater than  $q$ , then reducing it modulo  $q$  lowers the value.

77. That “reducing mod  $q$ ” does not require lowering a numerical value also finds support in the claim language. Claim 1[c] reads: “determining whether said output  $H(\text{SV})$  is less than said order  $q$  prior to reducing mod  $q$ .” The purpose of this step is to ensure that any reduction of  $H(\text{SV}) \bmod q$  (*i.e.*, computing “ $H(\text{SV}) \bmod q$ ”) must occur *after* the determination of whether  $H(\text{SV})$  is less than  $q$ . The claim goes on to reject  $H(\text{SV})$  and repeat the method if  $H(\text{SV})$  is not less than  $q$ , thus generating a new random number and a new  $H(\text{SV})$ . The claim, in other words, requires repeatedly generating a new  $\text{SV}$  and new  $H(\text{SV})$  until  $H(\text{SV}) < q$ .

78. Under the plain language of the claim, therefore, “reducing mod  $q$ ” would *never* result in a reduction of the value of  $H(\text{SV})$  – the operation “ $H(\text{SV}) \bmod q$ ” would *always* result in the same value because  $H(\text{SV})$  is always less than  $q$  when  $H(\text{SV})$  is



accepted, and thus reduction mod  $q$  might occur.<sup>10</sup> Accordingly, the fact that the claim

---

<sup>10</sup> One might wonder why the challenged claims of the '961 patent recite “prior to reducing mod  $q$ ” when, as explained in the text, the claims are designed to ensure that “reducing mod  $q$ ” could not change the value of  $H(SV)$  when selected as a cryptographic key. The prosecution history suggests that “prior to reducing mod  $q$ ” was added to create a distinction with the key generation technique described in the DSS. The applicants amended claim 1 on October 30, 2007 to add “prior to reducing mod  $q$ ,” and explained that the DSA/DSS “requires the reduction of the integer mod  $q$  which can expect that more values will lie in the first interval than the second, hence the bias.” (Amendments to Claims and Remarks, October 30, 2007, filed in U.S. Appl. Ser. No. 10/025,924, Ex. 1020.) In any event, the amended claims would allow existing DSA/DSS implementations to continue to perform “mod  $q$ ” operations on an accepted value of  $H(SV)$  to more closely track the description in the DSS (Appendix 3), even though the acceptance-rejection technique in the claim

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

refers to this operation as “reducing mod  $q$ ” – even though the claim specifically is designed to guarantee that this operation will not result in a reduction in the value of  $H(SV)$  – provides compelling evidence that the phrase “reducing mod  $q$ ” is not limited to cases where a value is actually reduced.

79. Two of the prior art references that I have cited in this Declaration provide further support for my position that “reducing mod  $q$ ” simply refers to computing the remainder upon division by  $q$ . (*See* Schneier, Ex. 1008, p.242 (“The operation  $a \bmod n$  denotes the residue of  $a$ , such that the residue is some integer from 0 to  $n - 1$ . This operation is **modular reduction**.”) (italics and bold in original); Menezes, Ex. 1005, p.599 (“If  $z$  is any integer, then  $z \bmod m$  (the integer remainder in the range  $[0, m - 1]$  after  $z$  is divided by  $m$ ) is called the *modular reduction* of  $z$  with respect to modulus  $m$ .”) (italics in original).) Both of these references treat “modular reduction” as synonymous with computing the remainder upon division by a modulus, regardless of whether a

---

guarantees that the step of “reducing mod  $q$ ” could never alter the value of  $H(SV)$  when accepted and used as a cryptographic key.

numerical value is lowered. In my opinion, these references are far more authoritative in understanding what “reducing” means in the context of “reducing mod  $q$ ” than the non-technical dictionaries cited by Dr. Rubin. The term “reducing mod  $q$ ” thus simply refers to computing the remainder upon dividing a value by  $q$ .

## **VI. APPLICATION OF THE PRIOR ART TO THE CHALLENGED CLAIMS OF THE '961 PATENT**

80. I have reviewed and analyzed the prior art references and materials listed in **Section I.B** above. In my opinion, each limitation of claims 1, 2, 5, 15, 16, 19, 23, 24, and 27 is disclosed and rendered obvious by the prior art and the admissions in the specification of the '961 patent. I have identified the following grounds of unpatentability with respect to the challenged claims:

<b>Ground</b>	<b>Prior Art References</b>	<b>Claim(s)</b>
1	DSS (Ex. 1004), Schneier (Ex. 1008), and Rose (Ex. 1006)	1, 2, 5, 15, 16, 19
2	DSS, Schneier, and Menezes (Ex. 1005)	1, 2, 5, 15, 16, 19
3	DSS, Schneier, Rose, Rao (Ex. 1018), and Floyd (Ex. 1019)	23, 24, 27
4	DSS, Schneier, Menezes, Rao, and Floyd	23, 24, 27

81. **Grounds 3-4** largely mirror **Grounds 1-2** but add Rao and Floyd solely

to address the “arithmetic processor” limitation recited in independent claim 23.

**Grounds 3-4** otherwise rely on the same mapping of the DSS, Schneier, Rose, and Menezes references as **Grounds 1-2**.

82. Counsel has informed me that DSS, Schneier, Rose, Menezes, Rao, and Floyd qualify as prior art to the '961 patent at least because each of them was published before December 27, 2000, the earliest application filing date appearing on the face of the '961 patent. I will provide a brief summary of each reference before applying the prior art to the challenged claims.

**B. Brief Description and Summary of the Prior Art**

**1. Brief Summary of DSS (EX. 1004)**

83. **DSS**, the Digital Signature Standard (DSS), describes the Digital Signature Algorithm (DSA), an algorithm published and adopted as a standard by the National Institute of Standards and Technology (“NIST”), which is division of the United States Department of Commerce. For clarity, the acronym “DSA” refers to the Digital Signature Algorithm, and “DSS” to the publication that describes it.

84. DSS explains that “[d]igital signatures are used to detect unauthorized modifications to data and to authenticate the identity of the signatory.” (DSS, Ex. 1004, Abstract.) Exhibit 1004 contains the 1994 version of DSS (FIPS 186), an

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

earlier version of the DSS standard (FIPS 186-2) specifically referenced in the '961 patent specification.<sup>11</sup> ('961, 2:36-44.) As explained in DSS:

This Standard specifies a Digital Signature Algorithm (DSA) appropriate for applications requiring a digital, rather than written, signature. The DSA digital signature is a pair of large numbers represented in a computer as strings of binary digits. The digital signature is computed using a set of rules (i.e., the DSA) and a set of parameters such that the identity of the signatory and integrity of the data can be verified. The DSA provides the capability to generate and verify signatures.

(DSS, Ex. 1004, p.1, Explanation.) Because it was adopted as a standard by the United States government, the DSA as described in the DSS is one of the most well-known and influential cryptographic techniques ever created. Persons of ordinary skill in the art would have been familiar with the DSA and would likely have read

---

<sup>11</sup> As explained earlier, I will cite primarily to the first version of DSS (FIPS 186/1994). (DSS, Ex. 1004.) The techniques for generating the ephemeral key **k** in DSS did not materially change from FIPS 186 to FIPS 186-2, and as such, the description provided in the '961 patent mirrors the techniques set forth in Exhibit 1004.

the DSS (or another publication describing it) in the course of their research and development of cryptographic systems.

85. This Declaration focuses on a very small portion of the DSS describing the process of generating a particular cryptographic key,  $\mathbf{k}$  – which is relevant to the subject matter of the challenged claims. Appendix 3 of the DSS describes the process of generating  $\mathbf{k}$  in detail, and Appendix 5 provides a concrete example using real values. (DSS, Ex. 1004, pp.13-16 (App. §3.2), 18-20.) As explained in Appendix 3.2:

**3.2. ALGORITHM FOR PRECOMPUTING ONE OR MORE  $\mathbf{k}$  AND  $\mathbf{r}$  VALUES**

This algorithm can be used to precompute  $\mathbf{k}$ ,  $\mathbf{k}^{-1}$ , and  $\mathbf{r}$  for  $m$  messages at a time. Algorithm:

Step 1. Choose a secret initial value for the seed-key, KKEY.

Step 2. In hexadecimal notation let

$t = \text{EFCDAB89 98BADCFE 10325476 C3D2E1F0 67452301}.$

This is a cyclic shift of the initial value for  $H_0 \parallel H_1 \parallel H_2 \parallel H_3 \parallel H_4$  in the SHS.

Step 3. For  $j = 0$  to  $m - 1$  do

a.  $\mathbf{k} = G(t, \text{KKEY}) \bmod q.$

b. Compute  $\mathbf{k}_j^{-1} = \mathbf{k}^{-1} \bmod q.$

c. Compute  $\mathbf{r}_j = (g^{\mathbf{k}} \bmod p) \bmod q.$

d.  $\text{KKEY} = (1 + \text{KKEY} + \mathbf{k}) \bmod 2^b.$

(DSS, Ex. 1004, p.14, App. §3.2 (highlighting added).) Most of the steps shown in Appendix 3.2 (including the ones that were omitted from the excerpt above) describe features that are not recited in the challenged claims, such as generating keys for multiple messages. The two yellow highlighted steps of Appendix 3.2 represent the steps actually reflected in the challenged claims.

86. “**Step 1**” states that to compute  $k$ , one must “[c]hoose a secret initial value for the seed-key, KKEY.” (DSS, Ex. 1004, p.14, App. §3.2.) This step corresponds to step [a] of claim 1, “generating a seed value  $SV$  from a random number generator.” The DSS explains that “[a]ny implementation of the DSA requires the ability to generate random or pseudorandom integers. Such numbers are used to derive a user’s private keys,  $x$ , and a user’s per message secret number,  $k$ .” (DSS, Ex. 1004, p.13.)

87. “**Step 2**” requires little discussion. It assigns a value to  $t$ , a constant used when hashing KKEY in Step 3(a) described below. Step 2 is not directly reflected in the challenged claims, but under the DSA, it is a necessary prerequisite to the particular hashing function used, as described below.

88. “**Step 3**,” subpart (a), states: “ $k = G(t, KKEY) \bmod q$ ” (DSS, Ex. 1004, p.14 (emphasis added).) In plain English, this step performs a hashing function on

the seed-key from Step 1 (KKEY), performs a “mod q” operation on the output of the hashing function, and then assigns the resulting value to k.

89. The DSS explains that “G(t,c)” refers to a one-way hashing function using, for example, the Secure Hash Algorithm (SHA). (DSS, Ex. 1004, p.13.) The term “hashing function” refers to a computational algorithm that uses input data to generate an output “hash” of a fixed length (*e.g.*, 160 bits). For example, if one were to feed the message “Have a nice day” into the SHA-1 hashing function, the function would output the following hash: b6bab9bc8870165ecb6c16e713339e4981432239 (in hexadecimal). (This result was computed using <http://www.sha1-online.com>.)<sup>12</sup>

---

<sup>12</sup> While the DSS (Ex. 1004) uses the “SHA” one-way hash function, the ’961 patent describes a later version of the DSS using the “SHA-1” one-way hash function. The “SHA-1” hash was developed as an improvement to the SHA hash function. Both “SHA” and “SHA-1” are one-way hash functions that generate 160-bit hashed values. Any differences between the hash functions is not relevant to my analysis and I do not discuss them further.



Most hashing functions, including the one specified in the DSS, are “one-way” in that the hashing function can be used to generate a hash from an input value, but it is not possible to recover the input value from the hash. Well-designed cryptographic hashing functions are often used to create unique identifiers because it is exceedingly unlikely that two different inputs could generate the same hash.

90. The “t” in the “G(t,c)” hashing function refers to the value specified in Step 2, and “c” refers to the seed value on which the hash operates. G(t,KKEY) therefore generates a hash of the seed-key, KKEY, from Step 1. This corresponds to step [b] of claim 1, “performing a hash function H( ) on said seed value to provide an output H(SV).”

91. Step 3(a) also specifies that a “mod q” operation is performed on the hash, *i.e.*, “**k = G(t,KKEY) mod q**.” (DSS, Ex. 1004, p.14, App. §3.2 (emphases added).) As explained previously, the purpose of this “mod q” operation is to ensure that the result falls within the desired range, *i.e.*, from 0 to **q** minus 1.

92. It is at this point that the ’961 patent steps in and augments the key generation process described in the DSS. Claim 1[c] recites the step of “determining whether said output H(SV) [*i.e.*, G(t, KKEY) in DSS] is less than said order q prior to reducing mod q.” The claim therefore requires that the claimed determination

take *place* before any reduction modulo  $q$  takes place. The DSS does not disclose the determining step, but the recited determination – and all the subsequent steps involved in accepting and rejecting an output  $H(SV)$  – would have been obvious over DSS in view of the other references described below.

## **2. Brief Summary of Rose (EX. 1006)**

93. **Rose** is a newsgroup article dated March 10, 1993, authored by Greg Rose, with Subject *Re: “Card-shuffling” algorithms*. Rose posted to two newsgroups, sci.crypt and sci.math, on a global service known as USENET.

94. USENET refers to a global communications system for reading and disseminating messages (often called “articles” or “posts”) to users through the Internet. USENET as of December 2000 (and continuing today) allowed users to browse and view articles and reply to articles posted by others. USENET was widely available to computer users throughout the world and was regarded as a useful resource for researchers and professionals to share technical information with others in the field. In fact, some of the most famous technologies we take for granted today were first introduced to the world through USENET articles. For example, Tim Berners-Lee posted a USENET article in 1991 to introduce and describe the World Wide Web project, Lunus Torvalds posted a USENET article to announce the first

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

version of Linux,<sup>13</sup> and Jeff Bezos posted a USENET article in 1994 seeking Unix developers to join a start-up company that would later become Amazon.com.<sup>14</sup> As of 1994, USENET had more than 2.7 million users, and more than 99,000 sites that hosted USENET content. (See Jenny A. Fristrup, *USENET: Netnews for Everyone* (1994), Ex. 1014, p.10; *see also id.*, p.26.)

95. Discussions on USENET are organized into a series of newsgroups identified by a hierarchical naming structure. (Frstrup, Ex. 1014, p.18.) For example, the strings “**sci.crypt**” and “**sci.math**” appearing in Rose identify the newsgroups to which the article was posted. The “sci” refers to the top-level hierarchy, science. “Issues of a scientific nature are discussed in this branch of the USENET newsgroup tree.” (Frstrup, Ex. 1014, p.170; *see also id.*, p.384 (“**sci\*** The

---

<sup>13</sup> See Famous Usenet Posts – WWW, Linux, Netscape (May 19, 2017), available at <<http://www.ngrblog.com/famous-usenet-posts/>> (last visited March 13, 2019).

<sup>14</sup> See also <<https://www.businessinsider.com/amazon-first-job-listing-posted-by-jeff-bezos-24-years-ago-2018-8>> (last visited March 13, 2019).

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

set of USENET newsgroups devoted to the discussion of things related to scientific issues.”) (bold in original.) As of 1994, there were dozens of newsgroups within “sci,” and “sci.crypt” focused on discussions relating to “[d]ifferent methods of en/decryption.” (Fristrup, Ex. 1014, p.174.)

96. Users could read and post USENET articles using widely available software programs known as “newsreaders” that, among other things, allowed users to select from available USENET newsgroups, locate and read articles within those newsgroups, and post new articles or reply to existing articles. (Fristrup, Ex. 1014, pp.360-363.) One common newsreader was known as “Read News” (or “rn”), a text-based newsreader that was included with many UNIX operating system distributions.

97. The “sci.crypt” newsgroup on which Exhibit 1006 was posted was regularly trafficked by researchers and programmers interested in cryptography. I personally read and posted articles on USENET beginning in the late 1990s, and on several occasions posted articles to the “sci.crypt” newsgroup. In my opinion, based on my own personal experience with USENET and with the newsgroup sci.crypt, the Rose article would have been publicly accessible to persons of ordinary skill in the art in cryptography well before December 2000.

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

98. Rose responds to a previous posting that had suggested generating random numbers from 0-2 using the following line of code (in the C programming language): “`r = lrand48() % 3`”.

99. This code, as explained previously, obtains a random number from a random number generator (`lrnd48()`), and then performs a “mod 3” operation on it to ensure that the resulting random number ranges from 0-2. Rose explains that, among other problems, this method of generating random numbers causes a modulo bias problem:

Technically, there IS a problem with the above generator. The example above demonstrates it. When the desired interval doesn't exactly divide the interval you already have, the smaller numbers are more likely than the larger ones. What you really have to do is throw away any result from the original generator that is too big.

(Rose, Ex. 1006, p.2.) Rose, like the Brecher and Jackson USENET articles described in **Part IV.D**, then sets forth source code for a random number generator that determines whether a generated number is greater than or equal to a given bound (“biggest”). If so, the code will reject the random number and generate another:

```

/*
 * Roll a random number [0..n-1].
 * Avoid the slight bias to low numbers.
 */
int
roll(int n)
{
    long    rval;
    long    biggest;
#define BIG 0x7FFFFFFFL /* biggest value returned by random() */

    if (n == 0)
        return 0;
    biggest = (BIG / n) * n;
    do {
        rval = random();
    } while (rval >= biggest);
    return rval % n;
}

```

Comment to the code explaining that this code will generate a random number in the range from 0 to n-1, and avoid the modulo “bias”

Define function roll() which takes input “n” defining the desired range for random numbers (from 0 to n-1)

**BIG** identifies the largest number the underlying random number generator (“random()”) may output

**Biggest** is assigned to the largest multiple of “n” that is less than or equal to **BIG** (the “/” function returns the number of times **BIG** can be divided by n ignoring the remainder, i.e.,  $4/3 = 1$ )

This is a loop that (1) generates a random number **rval** using a random number generator, (2) compares **rval** against **biggest**. If **rval** is greater than or equal to **biggest**, **rval** is rejected and the loop continues and generates another **rval**

At this point, the computer has broken free from the loop because **rval** is less than **biggest**. The last step is to compute “**rval mod n**”, to guarantee that the returned number falls within the desired range from 0 to n-1. This introduces no modulo bias because “biggest” (which was used in the loop above) was selected as a whole number multiple of “n”.

(Rose, Ex. 1006, p.2 (annotations added).)

100. Rose thus discloses a simplified rejection method for avoiding modulo bias in random number generation. The description in Rose renders the challenged claims obvious in view of the teachings of DSS.

### 3. Brief Summary of Menezes (EX. 1005)

101. Menezes, entitled *Handbook of Applied Cryptography* (1997), is a textbook “intended as a reference for professional cryptographers, presenting the

techniques and algorithms of greatest interest to the current practitioner, along with supporting motivation and background material.” (Menezes, Ex. 1005, p.xxiii (Preface).) Two of its authors (Alfred J. Menezes and Scott A. Vanstone) were named inventors on the '961 patent, but there is no evidence that the book was ever cited or considered by the Patent Office during the prosecution of the '961 patent. I know from my personal experience that Menezes was publicly accessible by the late 1990s, as I personally studied a copy during that period.

102. Like Rose, Menezes discloses the ability to generate uniformly distributed random numbers by rejecting any numbers that fall outside a desired range. Menezes explains that the security of many cryptographic systems depends upon the ability to generate random or unpredictable sequences. (Menezes, Ex. 1005, p.169, §5.1.) Menezes explains that “[a] *random bit generator* is a device or algorithm which outputs a sequence of statistically independent and unbiased binary digits.” (Menezes, Ex. 1005, p.170, §5.1 (italics in original).) Menezes describes how to employ this technique to generate a random number between 0 and  $n$ :

A random bit generator can be used to generate (uniformly distributed) random numbers. For example, a random integer in the interval  $[0, n]$  can be obtained by generating a random bit sequence of length  $\lfloor \lg n \rfloor +$

1, and converting it to an integer; if the resulting integer exceeds  $n$ , one option is to discard it and generate a new random bit sequence.

(Menezes, Ex. 1005, p.170, §5.2 (italics in original; underlining added).) Like Rose, therefore, Menezes discloses the purported point of novelty of the '961 patent – the ability to obtain random numbers within a desired range by rejecting random numbers that fall outside that range. Menezes thus renders the challenged claims obvious in view of the teachings of DSS.

#### **4. Brief Summary of Schneier (EX. 1008)**

103. **Schneier**, entitled *Applied Cryptography* (2d ed. 1996), is a book that provides “both a lively introduction to the field of cryptography and a comprehensive reference.” (Schneier, Ex. 1008, p.xx (Preface).) Schneier describes various techniques relating to cryptography, including techniques for generating random and pseudorandom numbers. (Schneier, Ex. 1008, pp.421-428.) I know from my personal experience that Schneier was publicly accessible by the late 1990s, as I personally owned and studied from a copy during that period.

104. This Declaration cites Schneier for its description of true random numbers and their benefit over pseudorandom numbers in cryptographic key generation. As Schneier explains:



Sometimes cryptographically secure pseudo-random numbers are not good enough. Many times in cryptography, you want real random numbers. Key generation is a prime example. It's fine to generate random cryptographic keys based on a pseudo-random sequence generator, but if an adversary gets a copy of that generator and the master key, the adversary can create the same keys and break your cryptosystem, no matter how secure your algorithms are. A random-sequence generator's sequences cannot be reproduced. No one, not even you, can reproduce the bit sequence out of those generators.

(Schneier, Ex. 1008, pp.421-22, §17.14 (underlining added).) I have cited Schneier in the event it is argued that the term “random number generator,” as recited in claim 1[a], requires a true (as opposed to pseudo) random number generator.

## **5. Brief Summary of Rao (EX. 1018) and Floyd (EX. 1019)**

105. I have cited Rao and Floyd solely in **Grounds 3-4** in connection with a limitation in independent claim 23, which recites a cryptographic unit comprising “**an arithmetic processor**” for performing the same steps recited in independent claim 1. The specification of the '961 patent devotes only one sentence to the “arithmetic processor,” and it is depicted as a nondescript box in Figure 1. (See '961, 3:33-38 (“Each of the correspondents **12**, **14** includes a secure cryptographic function **20** including a secure memory **22**, an arithmetic processor **24** for

performing finite field operations, a random number generator **26** and a cryptographic hash function **28** for performing a secure cryptographic hash such as SHA-1.”), Fig. 1 (item 24).) DSS, Schneier, Rose, and Menezes do not appear to expressly disclose the use of “an arithmetic processor,” as claimed, but such a processor would have been obvious to a person of ordinary skill in the art.

106. I have cited Rao and Floyd, which provide information about computer architecture and organization that would have formed the basic background knowledge of a person of ordinary skill in the art. These references confirm that the claimed “**arithmetic processor**” was not an invention of the ’961 patent. An “arithmetic processor” instead merely describes the processing functionality built-in to commercially available processors available long before December 2000.

107. **Rao** is a 1974 textbook that explains how digital computers are divided up into functional units or subsystems, including an “arithmetic processor” (AP). (Thammavarapu R. N. Rao, *Error Coding for Arithmetic Processors* (1974), Ex. 1018, p.39.) Rao explains that “[a]n arithmetic processor is the part of a computer that provides the logic circuits required to perform arithmetic operations such as ADD, SUBTRACT, MULTIPLY, DIVIDE, SQUARE-ROOT, etc.” (Rao, Ex. 1018, p.41.) **Floyd** explains that in a more modern computer that includes a

microprocessor, the arithmetic processing is performed by a portion of a computer's central processing unit (CPU) known as an arithmetic/logic unit (ALU). (Nancy A. Floyd, *Essentials of Data Processing* (1987), Ex. 1019, pp.45-46.) "The arithmetic/logic unit (ALU), as the name implies, contains all of the circuitry necessary to perform the computer's arithmetic and comparison operations." (*Id.*) A person of ordinary skill in the art would have recognized that an "arithmetic processor" was a built-in feature of substantially all computing devices as of December 2000, and therefore does not provide a non-obvious distinction.

**C. Ground 1: Comparison of the Prior Art to the Challenged Claims Based on DSS, Schneier, and Rose**

**1. Independent Claim 1**

- a. "A method of generating a key  $k$  for use in a cryptographic function performed over a group of order  $q$ , said method including the steps of:" Claim 1[Pre]**

108. To the extent the preamble of claim 1 recites a limitation to the claim, it is admitted prior art in the '961 specification and disclosed by DSS.

109. As I explained in **Part III.A** above, I understand that admissions by the patent application regarding the presence of features in the prior art, whether contained in the patent specification or made during the prosecution history, can be

relied upon to show anticipation or obviousness of a claim. In my opinion, the “Background of the Invention” of the ’961 patent provides those types of admissions, and unequivocally acknowledges that the subject matter of the preamble was part of the well-known Digital Signature Algorithm (DSA):

Some of the more popular protocols for signature are the ElGamal family of signature schemes such as the Digital Signature Algorithm or DSA. The DSA algorithm utilizes both long term and ephemeral keys to generate a signature of the message.

\* \* \*

The DSA requires the signatory to select an ephemeral key  $k$  lying between 1 and  $q-1$ .

(’961, 1:52-56, 1:63-64 (underlining added).) The ’961 specification goes on to describe in more detail how DSS describes calculation of ephemeral key  $k$ :

One such implementation [of DSA] is the DSS mandated by the National Institute of Standards and Technology (NIST) FIPS 186-2 Standard. The DSS stipulates the manner in which an integer is to be selected for use as a private key. A seed value, SV, is generated from a random number generator which is then hashed by a SHA-1 hash function to yield a bit string of predetermined length, typically 160 bits. The bit string represents an integer between 0 and  $2^{160}-1$ . However this

integer could be greater than the prime  $q$  and so the DSS requires the reduction of the integer mod  $q$ , i.e.  $k = \text{SHA-1}(\text{seed}) \bmod q$ .

(’961, 2:36-46 (underlining added).) The passage discloses the computation of the key  $k$  because it discloses that “ $k = \text{SHA-1}(\text{seed}) \bmod q$ ” (’961, 2:46), which confirms that the key  $k$  is being computed. The passage also confirms that the value  $k$  in DSA “is to be selected for use as a private key” (’961, 2:38-40). Specifically, the specification explains key  $k$  is an “ephemeral private key,” which can be used to generate the signature on a message. (’961, 1:26-47.) The patent specification therefore admits that the DSA/DSS disclosed “**generating a key  $k$  for use in a cryptographic function.**”

110. The Background of the Invention of the ’961 patent further explains how the ephemeral key  $k$  is directly used to calculate the two signature components for a particular message  $m$  in DSA: “A first signature component  $r$  is generated from the generator  $\alpha$  such that  $r = (\alpha^k \bmod p) \bmod q \dots$  A second signature component  $s$  is generated such that  $s = k^{-1}(H(m) + dr) \bmod q \dots$ ” (’961, 1:64-67.) Both of these formulae use the value of  $k$ . And the generation of these two signature components clearly qualifies as **using  $k$  “in a cryptographic function,”** as claimed, because the values of  $r$  and  $s$  determine the signature for the message. (’961, 2:1 (“The signature

on the message  $m$  is  $(r, s)$ .”.)

111. The '961 specification also confirms that the prior art DSA technique generates a key  $k$  for use in a cryptographic function “**performed over a group of order  $q$ ,**” as recited. The term “**group of order  $q$** ” refers to a set satisfying certain algebraic properties and containing  $q$  elements. (*See, e.g., Merriam-Webster’s Collegiate Dictionary* (10th ed. 1998), Ex. 1028, p.515 (defining “group” as “a mathematical set that is closed under a binary associative operation, contains an identity element, and has an inverse for every element”), p.818 (defining “order” as “the number of elements in a finite mathematical group”); *see also id.*, p.1072 (defining “set” as “a collection of elements and esp. mathematical ones (as numbers or points)”); *The American Heritage Dictionary* (4th ed. 2000), Ex. 1029, p.776 (defining “group” as “[a] set with a binary associative operation such that the operation admits an identity element and each element of the set has an inverse element for the operation”), p.1238 (defining “order” as “[t]he number of elements in a finite group”); *see also id.*, p.1593 (defining “set” as “[a] collection of distinct elements having specific common properties: *a set of positive integers*”) (italics in original).) In the context of the '961 patent, the “group of order  $q$ ” corresponds to

the set of integer values between 0 and  $q$  minus 1, i.e.,  $\{0, 1, 2, \dots, q-1\}$ . ”).<sup>15</sup> Such a set has  $q$  elements and satisfies the properties of a group with the associated binary operation of addition modulo  $q$ , and is thus a “**group of order  $q$ .**”

112. The claim requirement for “a key  $k$  for use in a cryptographic function performed over a group of order  $q$ ” is satisfied by the DSS, because the signature  $(r, s)$  generated by the cryptographic function consists of two values  $(r$  and  $s)$  that are elements of the “group of order  $q$ ,” i.e., the set of integers between 0 and  $q$  minus 1.

---

<sup>15</sup> As discussed previously, the value of  $q$  in DSA is a preselected prime number. (’961, 1:54-62 (“The DSA algorithm utilizes both long term and ephemeral [private] keys to generate a signature of the message. The DSA domain parameters are preselected. They consist of... a prime number  $q$  of a predetermined bit length....”) (underlining added), 2:44-46; *see also id.*, 3:66-4:2, Claims 5, 19, 27.) Thus, to provide a simple example by way of illustration, if the value of “ $q$ ” is set to the prime number 3, then the “group of order  $q$ ” would be the set of numerical values  $\{0, 1, 2\}$ .

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

This is because, as the '961 patent explains, the formulas for generating  $r$  and  $s$  are both performed using modular reduction based on modulus  $q$ . ('961, 1:64-67 (“A first signature component  $r$  is generated from the generator  $\alpha$  such that  $r = (\alpha^k \bmod p) \bmod q$ ... A second signature component  $s$  is generated such that  $s = k^{-1}(H(m) + dr) \bmod q$ ....”) (emphases added).) The cryptographic signature values  $r$  and  $s$  output by these functions thus result in integer values between 0 and  $q$  minus one, because the “mod  $q$ ” operations in each function ensure that range of values. The DSA/DSS, admitted prior art in the Background of the '961 patent, therefore discloses “generating a key  $k$  for use in a cryptographic function performed over a group of order  $q$ .”

113. Even without reference to the admissions in the '961 patent, the preamble is also disclosed by DSS, which discloses the same technique described in the '961 patent specification. The step of “**generating a key  $k$  for use in a cryptographic function**” is shown in Appendix 3.2:



### 3.2. ALGORITHM FOR PRECOMPUTING ONE OR MORE $k$ AND $r$ VALUES

This algorithm can be used to precompute  $k$ ,  $k^{-1}$ , and  $r$  for  $m$  messages at a time. Algorithm:

Step 1. Choose a secret initial value for the seed-key, KKEY.

Step 2. In hexadecimal notation let

$t = \text{EFCDAB89 98BADCFE 10325476 C3D2E1F0 67452301}.$

This is a cyclic shift of the initial value for  $H_0 \parallel H_1 \parallel H_2 \parallel H_3 \parallel H_4$  in the SHS.

Step 3. For  $j = 0$  to  $m - 1$  do

a.  $k = G(t, \text{KKEY}) \bmod q.$

b. Compute  $k_j^{-1} = k^{-1} \bmod q.$

c. Compute  $r_j = (g^k \bmod p) \bmod q.$

d.  $\text{KKEY} = (1 + \text{KKEY} + k) \bmod 2^b.$

(DSS, Ex. 1004, p.14, App. §3.2 (highlighting added).) The claimed generation of  $k$  is shown in Step 3(a), shown in highlighting above, which assigns  $k$  the value of the hash of KKEY (*i.e.*,  $G(t, \text{KKEY})$ ),  $\bmod q$ . Appendix 5 of DSS further discloses that  $k$  is used to generate the signature components  $r$  and  $s$ :

### 5. SIGNATURE GENERATION

The signature of a message  $M$  is the pair of numbers  $r$  and  $s$  computed according to the equations below:

$$r = (g^k \bmod p) \bmod q \quad \text{and}$$

$$s = (k^{-1}(\text{SHA}(M) + xr)) \bmod q.$$

(DSS, Ex. 1004, p.6, §5 (highlighting added).) As shown in yellow, the generated key  $k$  is used in the formulas in Section 5 for computation of signature components

**r** and **s** (“**for use in a cryptographic function**”). Those computations also include a “mod  $q$ ” operation in blue to ensure that the signature values result in integers in the range of 0 to  $q$  minus 1, and thus the resulting values for **r** and **s** are integers lying in the range from 0 to  $q$  minus 1 (“**performed over a group of order  $q$** ”). DSS therefore discloses the preamble for the same reasons as the admissions in the ’961 patent described above.

**b. “generating a seed value SV from a random number generator” (Claim 1[a])**

114. As noted in the discussion of the preamble, the ’961 patent specification admits that this step was also disclosed by the DSS:

One such implementation [of DSA] is the DSS mandated by the National Institute of Standards and Technology (NIST) FIPS 186-2 Standard. The DSS stipulates the manner in which an integer is to be selected for use as a private key. A seed value, SV, is generated from a random number generator which is then hashed by a SHA-1 hash function to yield a bit string of predetermined length, typically 160 bits. The bit string represents an integer between 0 and  $2^{160}-1$ . However this integer could be greater than the prime  $q$  and so the DSS requires the reduction of the integer mod  $q$ , i.e.  $k = \text{SHA-1}(\text{seed}) \bmod q$ .

(’961, 2:36-46 (underlining added).) The ’961 patent therefore admits that the DSS disclosed “**generating a seed value SV from a random number generator.**”

115. This is confirmed by DSS itself. Appendix 3.2 identifies **Step 1** of the algorithm for computing the value of **k** as: “Choose a secret initial value for the seed-key, KKEY.” (DSS, Ex. 1004, p.14, App. §3.2.) The claimed “**seed value SV**” thus takes the form of the seed-key, “KKEY,” in DSS.

116. I note that Step 1 in DSS does not state that KKEY is generated from a random number generator, but this would have been obvious based on other disclosures in DSS. For example, the first sentences of Appendix 3 state that “[a]ny implementation of the DSA requires the ability to generate random or pseudorandom integers. Such numbers are used to derive a user’s private key, x, and a user’s per message secret number, k.” (DSS, Ex. 1004, p.13, App. §3 (underlining added).) It would have been apparent and obvious to a person of ordinary skill in the art that a random number “used to derive... a user’s per message secret number, k” (*id.*) refers to the value of the seed-key, KKEY in **Step 1** of the algorithm for computing **k**. This is because Step 1 is the only step in the computation of **k** in which a random value can be input into the process. The remaining steps of computing **k** simply act upon the value of KKEY and thus do not independently generate random numbers or consult a source of randomness. More specifically, the formula for calculating **k** is described in DSS as “ $k = G(t, KKEY) \bmod q$ ” (DSS, Ex. 1004, p.14, App. §3.2), and

the other variables in this formula aside from KKEY (*i.e.*,  $t$  and  $q$ ) are fixed and non-random. It therefore would have been obvious that the generation of a random number as disclosed in Appendix 3 is for “[c]hoos[ing] a secret initial value for the seed-key, KKEY.” (DSS, Ex. 1004, p.14, App. §3.2.)

117. I also note that DSS states that any implementation of the DSA “requires the ability to generate random or pseudorandom integers.” (DSS, Ex. 1004, p.13, App. §3 (underlining added).) It is unclear if the claimed “**random number generator**” recited in claim 1[a] (1) includes both random and pseudorandom number generators or (2) includes only real random number generators. As I will explain below, the prior art discloses and renders obvious claim 1[a] under either interpretation.

118. By way of background, a pseudorandom number generator (PRNG) generates numbers using a deterministic function applied to some initial input value. This produces output that appears random. But if one learns the input value and knows the PRNG algorithm, one could regenerate the random-looking output. For our purposes, a “true” or “real” random number generator, on the other hand, relies on a source of unpredictability or randomness (such as random noise) to generate true random output. (Schneier, Ex. 1008, pp.421-424, §17.14.) As explained in

**Part IV.B**, random number generators can use a variety of external sources of randomness to generate random values.

119. With respect to the '961 patent, I see nothing in the specification to suggest that the applicants intended to limit “**random number generator**” in claim 1[a] to only real (and not pseudo) random number generators. To the contrary, the '961 patent specifically cites the DSA/DSS and states that in DSS, “[a] seed value, SV, is generated from a random number generator” ('961, 2:40-41 (underlining added)). DSS specifically allows that seed value to be generated either randomly or pseudorandomly. (*See* DSS, Ex. 1004, p.13, App. §3 (“Any implementation of the DSA requires the ability to generate random or pseudorandom integers.”) (underlining added); *see also* Federal Information Processing (FIPS) Publication 186-2, *Digital Signature Standard (DSS)* (January 27, 2000), Ex. 1015, p.16 (“Any implementation of the DSA requires the ability to generate random or pseudorandom

integers.”) (underlining added).)<sup>16</sup> A person of ordinary skill in the art would thus have interpreted the ’961 patent’s reference to a “random number generator” as including both random and pseudorandom number generators, consistent with the DSA/DSS embodiment described in the ’961 patent.

120. In any event, because DSS clearly discloses the generation of “random or pseudorandom integers” (DSS, Ex. 1004, p.13, App. §3 (underlining added)), it renders obvious claim 1[a] even if the claim is construed narrowly. A person of ordinary skill in the art would have been motivated to use random (instead of pseudorandom) numbers because they can generate stronger cryptographic keys.

121. This is because real random number generators, unlike pseudorandom

---

<sup>16</sup> As demonstrated in the text, the statement from DSS about “requir[ing] the ability to generate random or pseudorandom integers” appears identically in the 1994 version that I cite for **Ground 1**, and in the 2000 version (FIPS 186-2) identified in the Background of the ’961 patent. (*Compare* DSS, Ex. 1004, p.13 *with* Ex. 1015, p.16.)

number generators, generate truly random numbers rather than numbers that merely appear random. As explained in Bruce Schneier, *Applied Cryptography* (2d ed. 1996), Ex. 1008, this provides the preferred way to generate random numbers for cryptographic keys:

Sometimes cryptographically secure pseudo-random numbers are not good enough. Many times in cryptography, you want real random numbers. **Key generation is a prime example.** It's fine to generate random cryptographic keys based on a pseudo-random sequence generator, but if an adversary gets a copy of that generator and the master key, the adversary can create the same keys and break your cryptosystem, no matter how secure your algorithms are. A random-sequence generator's sequences cannot be reproduced. No one, not even you, can reproduce the bit sequence out of those generators.

(Schneier, Ex. 1008, pp.421-22, §17.14 (emphases added).) Schneier goes on to describe several known techniques for generating real random numbers, including hardware techniques for harvesting random noise, using the computer's system clock and keyboard latency, among others. (Schneier, Ex. 1008, pp.423-425.) Schneier also describes a technique for distilling randomness from a variety of sources accessible to the computer, and creating a "pool or reservoir that applications can draw from as needed." (Schneier, Ex. 1008, pp. 426-428.) No matter which

technique is employed, a person of ordinary skill in the art would have been motivated to use a real (as opposed to pseudo) random number generator to achieve the beneficial purpose of obtaining stronger cryptographic keys.

122. ***Rationale and Motivation to Combine:*** It would have been obvious to combine the teachings of DSS with Schneier, with no change in their respective functions, predictably resulting in the technique for generating key **k** in DSS in which the seed-key, KKEY, was always generated with a real (and not pseudo) random number generator. DSS and Schneier are analogous references in the same field of cryptography and key generation. In fact, Schneier devotes an entire portion of one chapter to a lengthy discussion of the DSA/DSS. (Schneier, Ex. 1008, pp.483-494, §20.1.) A person of ordinary skill in the art would have naturally known of Schneier in implementing the techniques described in the DSS.

123. Schneier provides an express motivation to combine, as explained above, by explaining that “[s]ometimes cryptographically secure pseudo-random numbers are not good enough. Many times in cryptography, you want real random numbers. Key generation is a prime example.” (Schneier, Ex. 1008, p.421, §17.14 (underlining added).) A person of ordinary skill in the art would have been motivated to use a real random number for the seed-key, KKEY, in DSS in order to



create a stronger key for a digital signature. Indeed, Schneier's description of the DSA/DSS explains that "[e]ach signature requires a new value of *k*, and that value must be chosen randomly." (Schneier, Ex. 1008, p.492 (italics in original; underlining added).) Because the seed-key KKEY in the DSS is the *only* value that changes for each new value of *k*, a person of ordinary skill in the art would have understood the benefit of using a real random number generator to generate KKEY.

124. A person of ordinary skill in the art would have perceived no technological obstacles to this combination, and would have had a reasonable expectation of success. Techniques for generating "real" random numbers were known and readily available to persons of ordinary skill in the art. (Schneier, Ex. 1008, pp.423-26.) Schneier describes a broad array of approaches, from using dedicated hardware to measure random noise to more straightforward software techniques that rely on the timing of events within the computer. (*Id.*) No matter the technique employed, a person of ordinary skill in the art would have had a reasonable expectation of success in implementing the DSS process of generating KKEY using a real random number generator.

c. **“performing a hash function  $H()$  on said seed value SV to provide an output  $H(SV)$ ” (Claim 1[b])**

125. As noted in the discussion of the preamble, the '961 patent specification also admits that this step was disclosed by the DSS:

One such implementation [of DSA] is the DSS mandated by the National Institute of Standards and Technology (NIST) FIPS 186-2 Standard. The DSS stipulates the manner in which an integer is to be selected for use as a private key. A seed value, SV, is generated from a random number generator which is then hashed by a SHA-1 hash function to yield a bit string of predetermined length, typically 160 bits. The bit string represents an integer between 0 and  $2^{160}-1$ . However this integer could be greater than the prime  $q$  and so the DSS requires the reduction of the integer mod  $q$ , i.e.  $k = \text{SHA-1}(\text{seed}) \bmod q$ .

('961, 2:36-46 (emphases added).) The '961 patent therefore admits that the DSS disclosed **“performing a hash function  $H()$  on said seed value SV to provide an output  $H(SV)$ .”** The output  $H(SV)$  corresponds to  $\text{SHA-1}(\text{seed})$  (an integer between 0 and  $2^{160}-1$ ) in the passage above.

126. And the '961 specification is accurate on this point – the DSS does disclose this step. Appendix 3.2 of the DSS identifies **Step 3(a)** of the algorithm for computing the value of  $k$  as: “ $k = \text{G}(\text{t}, \text{KKEY}) \bmod q$ .” (DSS, Ex. 1004, p.14, App. §3.2 (underlining added).) The claimed **“ $H(SV)$ ”** thus takes the form of the hash of

KKEY,  $G(t, KKEY)$ , in DSS. The DSS explains that “ $G(t, c)$ ” refers to a one-way hashing function using, for example, the Secure Hash Algorithm (SHA). (DSS, Ex. 1004, p.13 (“The algorithms employ a one-way function  $G(t, c)$ .... One way to construct  $G$  is via the Secure Hash Algorithm (SHA)....”).)

127. As explained in the summary above, the term “hashing function” refers to a computational algorithm that uses input data to generate an output “hash” of a fixed length (*e.g.* 160 bits). In paragraph 89 above, I provide an example of such a hash. Most hashing functions, including the SHA specified in the DSS, are “one-way” in that one can use the hashing function to generate a hash from an input, but one cannot generate the input from the hash. Well-designed cryptographic hashing functions are often used to create unique identifiers because it is exceedingly unlikely that two different inputs could generate the same hash.

128. The “ $t$ ” in the “ $G(t, c)$ ” hashing function refers to a fixed 160-bit constant in **Step 2** of DSS. (DSS, Ex. 1004, p.13, Step 2; *see also id.*, p.14, Step 5 (setting  $t$  to a new value if the algorithm in Appendix 3.2 is used to generate multiple values of  $k$ ).) The “ $c$ ” refers to the input on which the hash operates.  $G(t, KKEY)$  therefore generates a hash of the seed-key, KKEY, from Step 1. Accordingly, DSS discloses “**performing a hash function  $H( )$  on said seed value SV to provide an**

output  $H(SV)$ ,” as recited in claim 1[b].

- d. “determining whether said output  $H(SV)$  is less than said order  $q$  prior to reducing mod  $q$ , accepting said output  $H(SV)$  for use as said key  $k$  if the value of said output  $H(SV)$  is less than said order  $q$ , rejecting said output  $H(SV)$  as said key if said value is not less than said order  $q$ , if said output  $H(SV)$  is rejected, repeating said method” (Claim 1[c]-1[f])

129. DSS does not disclose this claim limitation. As mentioned above, **Step 3(a)** of the algorithm for computing the value of  $k$  in DSS states: “ $k = G(t, KKEY) \bmod q$ .” (DSS, Ex. 1004, p.14, App. §3.2 (emphases added showing modular reduction of  $G(t, KKEY)$  to obtain the value  $k$ .) The ’961 patent similarly admits that “the DSS require[d] the reduction of the integer mod  $q$ , i.e.  $k = \text{SHA-1}(\text{seed}) \bmod q$ .” (’961, 2:44-46 (emphases added).)

130. DSS thus performs a modular reduction on the result of the hash  $G(t, KKEY)$  (“**said output  $H(SV)$** ”), without determining whether the output  $H(SV)$  is less than order  $q$ . As explained in great detail in **Part IV.C** and **Part V.A**, the “mod  $q$ ” operation guarantees that the resulting number is within the range 0 to  $q$  minus 1, but creates the possibility of introducing modulo bias (the numbers from 0 to  $q$  minus 1 may not be uniformly distributed) that the ’961 patent seeks to avoid.

131. But these limitations of claim 1 are obvious over DSS in view of **Rose**.

Because claim limitations 1[c] through 1[f] are very closely intertwined and disclosed by overlapping disclosures in the DSS and Rose, I will address them together to avoid needless repetition.

132. **Rose** discloses a software-based technique for generating a random number within a specified range. Rose is a USENET article that responded to a previous article that had suggested generating random numbers from 0-2 using the following line of code (in the C programming language): “`r = lrand48() % 3.`”

Rose explained this code introduces a modulo bias problem:

Technically, there IS a problem with the above generator. The example above demonstrates it. When the desired interval doesn't exactly divide the interval you already have, the smaller numbers are more likely than the larger ones. What you really have to do is throw away any result from the original generator that is too big.

(Rose, Ex. 1006, p.2.) Rose provides source code (in the C programming language) for a random number generator that determines whether a generated number is greater than or equal to a value (“biggest”) representing the largest multiple of the size of the desired range that is less than or equal to the largest number the underlying random number generator can output. The code, and my annotations explaining it, is set forth below:

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

```
/*  
 * Roll a random number [0..n-1].  
 * Avoid the slight bias to low numbers.  
 */  
  
int  
roll(int n) {  
    long    rval;  
    long    biggest;  
#define BIG 0x7FFFFFFFL /* biggest value returned by random() */  
  
    if (n == 0)  
        return 0;  
    biggest = (BIG / n) * n;  
    do {  
        rval = random();  
    } while (rval >= biggest);  
    return rval % n;  
}
```

Comment to the code explaining that the code below will generate a random number in the range from 0 to n-1, and avoid the modulo “bias”

Define function roll() which takes input “n”, where the desired range for random numbers generated by roll will be from 0 to n-1

**BIG** identifies the largest number the underlying random number generator (“random()”) may output

**biggest** is assigned to the largest multiple of “n” that is less than or equal to **BIG**.

This is a loop that (1) generates a random number **rval** using random number generator random(); (2) compares **rval** against **biggest**. If **rval** is greater than or equal to **biggest**, **rval** is rejected and the loop continues and generates another **rval**

At this point, the program has broken free from the loop because **rval** is less than **biggest**. The last step is to reduce rval modulo n (i.e., to compute **rval mod n**), to guarantee that the returned number falls within the desired range from 0 to n-1. This introduces no modulo bias because “biggest” (which was used in the loop above) was selected as a whole number multiple of “n”.

(Rose, Ex. 1006, p.2 (annotations added).)

133. The operation of the code above would have been straightforward to any person of ordinary skill in the art with a rudimentary understanding of programming. In the interest of completeness, I will go through the function line-by-line to ensure that I cover all of its functionality. In order to use the function **roll**, other code would call it and pass in, as a parameter, an integer value “**n**” that represents the number of integers in the desired range. Because the lowest random

number is zero, **roll** will return a random number from 0 to **n**-1, as indicated by the comment at the beginning of the code. If “n” is set to 20, for example, the roll will return a value from 0 to 19. From the standpoint of the claims, “n” is analogous to “q” in that both define the desired range for the resulting random value.

134. The first two lines of **roll** declare two integer variables, **rval** and **biggest**, which are used later in the **roll** function. They are declared as type “**long**” which refers to signed integers, typically 32-bits in length. (See Brian W. Kernighan & Dennis M. Ritchie, *The C Programming Language* (2d ed. 1988), Ex. 1016, p.36, §2.2.) Because these two lines simply declare the variables and do not actually assign any value to them, I will discuss these variables in more detail below.

135. The third line defines **BIG** as 0x7FFFFFFFL,<sup>17</sup> which is a hexadecimal (base 16) representation of 2,147,483,647, which is the largest positive value for a

---

<sup>17</sup> The “L” at the end of “0x7FFFFFFFL” is not part of the actual number; it is an indicator to the compiler that the value should be treated as a long (32-bit) value. (See Kernighan, Ex. 1016, p.37, §2.3.)

32-bit signed integer. This represents the largest number that the underlying random number generator employed in Rose could produce. (Rose, Ex. 1006, p. 2 (assuming that random() returns numbers “in the range  $[0, 2^{31}-1]$ ”).) I note that this line does not represent actual code that will be executed; “#define” is a standard directive in the C programming language to define a macro. In this case, #define BIG 0x7FFFFFFF simply tells the compiler to replace any instance of “BIG” appearing later in the code with a constant value 0x7FFFFFFF. (See Kernighan, Ex. 1016, pp.14-15, §1.4.)

136. The next two lines simply perform a check, “if (n == 0)” then “return 0” which, in plain English, means: if the value of “n” is equal to zero, then exit this function and zero will be the return value. For purposes of my analysis, these two lines are not particularly pertinent. The type of check is usually performed to avoid a “division by zero” error in which the software attempts to divide a value by zero. (As is well known from arithmetic, a division by zero results in an undefined result.) The next line of **roll**, as explained below, divides BIG by “n”, so the check here verifies that “n” is non-zero before using it as a divisor in a division operation.

137. The next line assigns a value to “**biggest**,” the long integer declared above. In plain English, “**biggest = (BIG / n) \* n**” computes the quotient when BIG

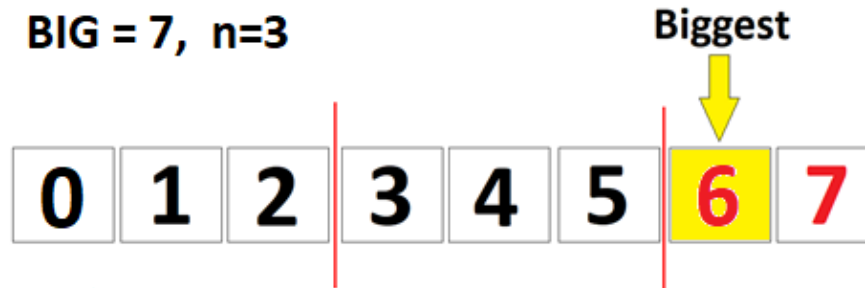


is divided by  $n$ , multiplies that result by  $n$ , and assigns the resulting value to “biggest.” (See Kernighan, Ex. 1016, p.41, §2.5.) The variable “biggest,” as a result of this calculation, represents the largest multiple of  $n$  that is less than or equal to BIG.

138. To use a highly simplified example, suppose “**BIG**” was 7, and “**n**” was 3. (That is, the biggest number that the underlying random number generator can produce is 7,<sup>18</sup> and the desired range of resulting random numbers is 0 to 2.) Because 7 is not evenly divisible by 3, the value of “**biggest**” will be 6 because  $(7 \div 3) \times 3 = 6$ . Although  $7 \div 3$  actually equals 2.733..., this line of code uses integer (whole number) math, so the  $7 \div 3$  operation here would return 2, the quotient (which is an integer value) of  $7 \div 3$ . The impact of this calculation is shown in the following diagram:

---

<sup>18</sup> I assume throughout that the underlying random number generator only produces nonnegative integers.



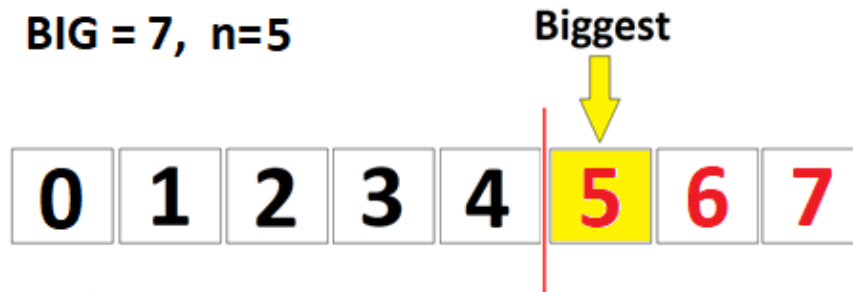
139. This is a critical step in Rose to eliminating the modulo bias discussed above because, as shown below, the code in Rose will only accept random numbers from the underlying generator that are less than “**biggest**” – in this case, in the range from 0 to 5.<sup>19</sup> This prevents modulo bias by preventing the random number generator (defined by the **roll** function) from using “residual” values in the range (such as the two values shown in red above) that might produce a non-uniform distribution.

140. But there is an even simpler example that is more applicable to DSS.

---

<sup>19</sup> In this respect, calling the variable “biggest” is somewhat of a misnomer – the code in Rose rejects a generated random number that is greater than or equal to “biggest.” A value equaling “biggest” thus represents a number that is too big.

Suppose “**BIG**” was 7, and “**n**” was 5. The value of “**biggest**” in this case would also be 5 (the same value as **n**) because  $(7 \div 5) \times 5 = 1 \times 5 = 5$ , using the integer math specified for the **roll** function.



In other words, “**biggest**” will equal “**n**” if the maximum value the underlying random number generator can produce is not at least twice as large as “**n**”. This is the case with DSS, as shown below, because the DSS specifies that “**H(SV)**” ( $G(t, \text{KKEY})$  in DSS) can never be at least twice as large as “**q**”.

141. The next three lines, “**do {**,” “**rval = random()**”, and “**} while (rval >= biggest)**” are the heart of the function and should be discussed together. These three lines establish what is known in the C programming language as a “do-while” loop, which instructs the computer to repeatedly “do” an operation or set of operations “while” a certain condition is true. (*See* Kernighan, Ex. 1016, pp.63-64, §3.6.) In this case, in plain English, these three lines do the following: (1) obtain a random

number by calling the “**random()**” function, and assign that number to the “**rval**” variable, and (2) check if “**rval**” is greater than or equal to “**biggest**,” described above; if so, go back to step (1) and get another random number. In other words, this code will continue obtaining a new random number and assigning it to “**rval**” until the value of “**rval**” is less than the value of “**biggest**.”

142. Using the simplified example above, if “**biggest**” was assigned 6, the do-while loop would continue obtaining random numbers by calling “**random()**” until it finally obtained one that was less than 6.<sup>20</sup> It is possible that an acceptable number could be obtained on the first call to **random()**, but in any event this loop

---

<sup>20</sup> The code checks to see if “**rval**” is greater than *or equal to* “**biggest**” – and not simply *greater than* – because the lowest random number that **roll** can return is zero (not one) as explained in the text.

will continue indefinitely until an acceptable number is obtained.<sup>21</sup>

143. As I will explain below, these lines, in combination with DSS, perform the step of “**determining whether said output H(SV) is less than said order q prior to reducing mod q.**” Here, “said output H(SV)” is analogous to **rval** in Rose, which is assigned the output of the underlying random number generator used by the **roll** function (*i.e.*, “**rval = random()**”). The term “**said order q,**” is satisfied by the value **q** as disclosed in the DSA/DSS, which is a prime number.<sup>22</sup> (*See* ’961, 1:54-62 (“The DSA algorithm utilizes both long term and ephemeral keys to generate a

---

<sup>21</sup> Although the do-while loop is designed to continue forever until it obtains a random number less than “**biggest,**” in practice it is unlikely that the loop would ever be executed more than a few times. In each iteration of the loop an acceptable number is obtained with probability at least half, and so the expected number of iterations of the loop until an acceptable number is obtained is at most two.

<sup>22</sup> This is consistent with dependent claim 5, which recites “wherein said order q is a prime number represented by a bit string of predetermined length L.”

signature of the message. The DSA domain parameters are preselected. They consist of... a prime number q of a predetermined bit length....”) (underlining added), 2:44-46; *see also id.*, 3:66-4:2.) The reference to “said order q” in the claim thus simply refers to the value of prime number **q** in DSS. (DSS, Ex. 1004, p.13, App. §3 (“These randomly or pseudorandomly generated integers are selected to be between 0 and the 160-bit prime q (as specified in the standard).”) (underlining added).)

144. And this (“said order q”) is analogous in Rose to “**biggest**” As noted below, in DSA/DSS,  $H(SV)$  is always less than twice the value of  $q$ . Thus, if “ $n$ ” in Rose is set to  $q$ , then the value “**biggest**” in Rose will hold the value of “ $n$ ” (the largest multiple of “ $n$ ” less than or equal to “**BIG**”), i.e.,  $q$ .

145. The step of “**determining**” under the combination of DSS and Rose thus takes place at the bottom of the loop with “**while (rval >= biggest)**”, which evaluates whether  $rval$  is less than **biggest**, and if so, terminates the do-while loop. More specifically, if  $rval$  is less than **biggest**, the next step is performed under the combination of DSS and Rose, “**accepting said output  $H(SV)$  for use as said key k if the value of said output  $H(SV)$  is less than said order q,**” as explained below. This is because if “**rval**” is less than “**biggest**,” the do-while loop terminates and the value of “**rval**” is not recalculated – it is instead accepted and used in the next line

of Rose (“return rval %n”).

146. And with respect to the step of “**rejecting said output H(SV) as said key if said value is not less than said order q; if said output H(SV) is rejected, repeating said method,**” this occurs under the combination of DSS and Rose because if rval is not less than biggest (*i.e.* rval is greater than or equal to biggest) – the do-while loop repeats and generates a new random number. I will explain in more detail below how Rose’s teachings can be adapted to DSS.

147. The final line of the **roll** function, “**return rval % n**”, exits the **roll** function by performing a modular reduction of “**rval**” and then returning the result. In this case, the percent sign (%) operator indicates a modulo operation (namely, computing **rval mod n**), which ensures that the random number returned is within the desired range (from 0 to n-1). (*See* Kernighan, Ex. 1016, p.41, §2.5.) Using the simplified example above, if “**n**” is set to 3, and the random number “**rval**” is 4 when exiting the loop, then **roll** will return 1, because  $4 \bmod 3 = 1$ . Similarly, if “**n**” is set to 6, but the random number “**rval**” was 4 when exiting the loop, then roll will return 4, because  $4 \bmod 6 = 4$ . Either way, the modular reduction ensures that the **roll** function does not return a random number outside the range specified by n (*i.e.*, 0 to n-1).

148. From the standpoint of the claims, this final line of code performs the same function as the claimed “**reducing mod q**,” by guaranteeing that whatever random number is generated, it will always fall within the desired range as discussed above. As noted, “**n**” in Rose is analogous to “**q**” of claim 1 in that both define the desired range for the random value. And as with the claims, the code at the bottom of the do-while loop in Rose determines that the random number is less than the upper bound of the desired range (or a multiple of the upper bound of the desired range) **prior to** performing the modular reduction.

149. As explained, the **roll** function described in Rose is designed to produce random numbers without “modulo bias.” Indeed, assuming the underlying random number generator functions properly (*i.e.*, generates uniformly distributed random numbers over its range), the **roll** function will eliminate modulo bias to the same extent as the technique described in the ’961 patent. This is because any numbers output by the underlying random number generator that could create a modulo bias – those that are not less than “biggest” – are rejected before any modular reduction is performed.

150. ***Manner, Rationale and Motivation to Combine:*** The **roll** function in Rose illustrates a simple and generic random number generation technique that could



readily be applied to any situation requiring the generation of random numbers within a specified range, including DSS. A person of ordinary skill in the art would have found it obvious to combine the techniques of Rose with the cryptographic key generation technique described in DSS. This combination would have predictably resulted in the technique of generating **k** in DSS enhanced with the random number generation technique in Rose, which would have resulted in the following sequence of steps:

- (1) Generate a (real or pseudo) random number for KKEY in accordance with DSS alone or in combination with Schneier (the claimed “**seed value SV**”), as explained for claim 1[a];
- (2) perform a hash of KKEY,  $G(t, KKEY)$ , according to DSS (the claimed “**output H(SV)**”), as explained for claim 1[b];
- (3) determine whether  $G(t, KKEY)$  (“**output H(SV)**”) is less than **q**, according to the teachings of Rose;
- (4) accept  $G(t, KKEY)$  (“**output H(SV)**”) if  $G(t, KKEY)$  **is** less than **q**, according to the teachings of Rose;
- (5) reject  $G(t, KKEY)$  if it is **not** less than **q**, by repeating the method and going back to step (1), according to the teachings of Rose; and
- (6) if  $G(t, KKEY)$  (“**output H(SV)**”) was accepted in step (4), perform a “mod **q**” operation on  $G(t, KKEY)$ , pursuant to the teachings of Rose, and provide it as a key **k** for use

in performing a cryptographic function, in accordance with DSS.

151. A person of ordinary skill in the art would have found this to be a trivially simple combination. In fact, there would be no need to compute “**biggest**” in the Rose function, because, based on the parameters used in DSS, the value of “**biggest**” will always equal **q**, *i.e.*, there is no case in which “BIG” (the maximum possible value output by the underlying random number generator) would ever be two or more times greater than **q**. This is because DSS requires the value of **q** to be a prime integer satisfying  $2^{159} < q < 2^{160}$ . (DSS, Ex. 1004, p.6, §4.) Thus, the smallest possible value of **q** is  $2^{159}+1$  (assuming this is a prime number). Because the output of the hashing function for KKEY in DSS (which corresponds to “**output H(SV)**”) is 160 bits long (DSS, Ex. 1004, p.13, App. §3; p.15, App. §3.3), the maximum hash value is  $2^{160}$  minus one. (*See also* ’961, 2:40-44 (explaining that in DSS, the hashed value – SHA-1(seed) – is “typically 160 bits” and “represents an integer between 0 and  $2^{160}-1$ ”).) Thus, the largest value of  $G(t, KKEY)$  (the claimed “**output H(SV)**”) is  $2^{160}$  minus 1, and that could never be two or more times larger than **q** (which has a minimum value of  $2^{159}+1$ ). Indeed, two times the minimum value of **q** equals  $2^{160}+2$ , which is larger than the maximum value of  $G(t, KKEY)$ .

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

So the step in Rose of calculating “**biggest**” could be skipped.

152. Accordingly, a pseudocode implementation of the combination of Rose and DSS might look like this:

```
generate_k (q)                // q is the max value
{
    do {

        // Get a number from real random number generator
        KKEY = [Output of random number generator];

        // Hash the random number
        H_KKEY = G(t, KKEY);

    } while (H_KKEY >= q); // Roll again unless
                        // H(KKEY) < q

    return H_KKEY mod q;    // Value for k
}
```

This exemplary pseudocode includes a do-while loop whose structure and logic follow closely from Rose, but the pseudocode uses values and techniques from the teachings of DSS and Schneier discussed above. This do-while loop uses the output of the underlying random number generator per the teachings of DSS and Schneier

to generate KKEY, instead of the random() function used in Rose.<sup>23</sup> It also includes the extra step of hashing the value of the random number KKEY — (G(t, KKEY)) — which is specified in DSS. (DSS, Ex. 1004, p.14, App. §3.2 (Step 3(a)).)

153. The pseudocode above also includes a final line that returns “H\_KKEY mod q” to be analogous to Rose, but in fact the modulo operation (*i.e.*, computing “H\_KKEY **mod** q”) is unnecessary here. Because H\_KKEY is always less than q when a value for H\_KKEY is accepted by the do loop, the operation “H\_KKEY mod q” will always just return H\_KKEY. In either case, because the modular reduction occurs after the do-while loop terminates, it does not violate the “prior to reducing

---

<sup>23</sup> As explained in the discussion of claim 1[a], to the extent the claimed “**random number generator**” refers to a real (and not pseudo) random number generator, this limitation is satisfied by the combination of DSS and Schneier as explained. Using the exemplary pseudocode in the text, therefore, each time a new random number is generated within the do-while loop it would be a real random number according to the teachings of DSS and Schneier, to the extent the claim so requires.

mod  $q$ ” requirement of claim 1[c].

154. A person of ordinary skill in the art would have had many reasons to make this combination. At the outset, DSS, Schneier, and Rose are analogous references in the field of random number generation and cryptography, and Rose contains disclosures reasonably pertinent to the problems to be solved in DSS – generating a random number for a cryptographic key that falls within a desired range. A person of ordinary skill in the art would have understood that Rose’s random number generation system could have been readily adapted to any system in which random number generation within a desired range was needed, and thus could readily have been adapted to DSS to provide a random number generation technique that avoids modulo bias. As explained above, the ability to generate random numbers is an integral part of the DSA/DSS. (DSS, Ex. 1004, p.13, App. §3 (“Any implementation of the DSA requires the ability to generate random or pseudorandom integers.”).) An improvement in random number generation would therefore have been pertinent to a person of ordinary skill in the art implementing DSS. A further sign of the analogous nature of the references is that Rose was posted to the “sci.crypt” newsgroup, a well-known USENET forum for discussing cryptographic techniques.

155. A person of ordinary skill in the art would have been motivated to combine in order to avoid the “modulo bias” problem described earlier, which is expressly described in Rose. As explained in **Part IV.C** above, the inventors on the ’961 patent were not the first to identify the modulo bias problem or to propose a potential solution. Rose’s comments preceding the “roll” function describe its purpose as “[a]void[ing] the slight bias to low numbers.” (Rose, Ex. 1006, p.2.) As further explained by Rose: “When the desired interval doesn’t exactly divide the interval you already have, the smaller numbers are more likely than the larger ones.” (*Id.*) Rose continues by providing the solution described in the ’961 patent: “What you really have to do is throw away any result from the original generator that is too big.” (*Id.*) By identifying the problem and the solution, Rose provides express motivations to adapt his technique for any application needing to generate random numbers with a specified range, such as DSS.

156. Rejecting values that fall outside a desired range was a well-known way of obtaining a uniform distribution of random numbers. (*See* Menezes, Ex. 1005, p.170, §5.2 (“A random bit generator can be used to generate (uniformly distributed) random numbers. For example, a random integer in the interval  $[0, n]$  can be obtained... if the resulting integer exceeds  $n$ , one option is to discard it and generate

a new random bit sequence.) (italics in original; underlining added).) At least two other programmers on the Internet – Brecher and Williams – independently made the same observation as Rose, and proposed solving it using a remarkably similar acceptance-rejection technique, as explained in **Part IV.C**. The technique described in Rose and recited in the '961 patent, in fact, is a highly simplified version of the rejection method described by John von Neumann in the early 1950s, as explained in **Part IV.C**. This was a concept well-known in computer science.

157. A person of ordinary skill would have been motivated to use Rose's technique to generate a key **k** in implementing the DSA described in the DSS. A person of ordinary skill in the art would have understood that modulo bias could, to some extent, weaken the strength of a cryptographic key by making some numbers more likely to occur than others. The '961 patent admits that it was known that the "mod q" operation in the DSA/DSS could create a bias towards lower values in the range of 0 to q-1. ('961, 2:53-55 ("With this algorithm, it is to be expected that more values will lie in the first interval than the second and therefore there is a potential bias in the selection of k.")) In fact, the '961 patent credits a third party, Daniel Bleichenbacher, with quantifying the potential impact of the modulo bias in the generation of k on the strength of the DSA/DSS. ('961, 2:56-61.) Accordingly, as

of the filing of the application for the '961 patent, the modulo bias problem for DSS was known and the motivation for avoiding it was clear. A person of ordinary skill in the art would thus have been motivated to use the technique in Rose to maintain a uniform distribution of random numbers over the range specified by  $q$ , resulting in a cryptographically stronger ephemeral key  $k$ , which would in turn have increased the strength of the resulting digital signature scheme.

158. The technique in Rose would also have been obvious to try. Rose itself confirms that there were a finite number of known and predictable solutions to the problem of obtaining random numbers within a desired range using a random number generator that outputs numbers in a larger range: (1) perform a modular reduction directly on the generated random number (which Rose criticizes), or (2) reject any generated number that falls outside the desired range (which Rose encourages). Both of these approaches were known, predictable solutions, and both would have provided a reasonable expectation of success.

159. Indeed, a person of ordinary skill in the art would have perceived no technological obstacle to this combination, and would have had every expectation of success in using Rose's technique. It is exceedingly simple, requiring only a few lines of "C" source code to implement. That code implements exceedingly basic



computational and comparison functions that would be understood to even beginning computer programmers. As explained above, Rose's technique as applied to DSS would have been even simpler than Rose itself, because neither the computation of "biggest" nor the modulo operation in **roll** would have been required. A person of ordinary skill in the art could have easily adapted Rose's technique to any other type of programming language or environment. Although Rose's exemplary source code handles random numbers in smaller ranges (*e.g.*, 32-bit values), programming techniques for handling and comparing large integer values (including the 160-bit numbers specified in the DSS) were well-known and easy to implement.

160. Finally, adapting the technique in Rose to the generation of the cryptographic key **k** in DSS would have had no significant impact on the performance of the system. Because the output of the hashing function ( $G(t, KKEY)$ ) in DSS (which corresponds to "**output H(SV)**") is 160 bits long (DSS, Ex. 1004, p.13, App. §3; p.15, App. §3.3), the maximum hash value is  $2^{160}$  minus one. But DSS requires the value of **q** (the desired range) to be a large prime number satisfying  $2^{159} < q < 2^{160}$ . (DSS, Ex. 1004, p.6, §4.) This means that under no circumstance could the hash of KKEY in the DSS (the claimed "**output H(SV)**") be two or more

times larger than  $q$ . This means that slightly more than half of the hashed values should be less than “ $q$ ” and thus be accepted. As a result, it is unlikely that the algorithm for generating  $k$ , adapted according to the teachings of Rose, would ever have to generate a new random number as part of the “do-while” loop more than a few times before obtaining a value less than  $q$ . It would have been trivial for a person of ordinary skill in the art to adapt an existing DSA/DSS implementation to generate another random value for KKEY, either by generating it on-the-fly using sources of randomness (*e.g.*, the hardware or software techniques described in Schneier), or drawing from a pool of random numbers that were generated beforehand from such random sources. The techniques described in Rose could be adapted to an implementation of the DSA/DSS using conventional and known programming techniques.

- e. **“if said output  $H(SV)$  is accepted, providing said key  $k$  for use in performing said cryptographic function, wherein said key  $k$  is equal to said output  $H(SV)$ .” (Claim 1[g])**

161. This step does not add anything that was not already addressed in the discussion of claim limitations 1[c] through 1[f] above. As explained, DSS and Schneier, in combination with Rose, disclose a key generation system that accepts

the hash of KKEY (“**said output  $H(SV)$** ”) if it is less than order  **$q$** .

162. Appendix 3.2 of the DSS identifies **Step 3(a)** of the algorithm for computing the value of  **$k$**  as: “ **$k = G(t, KKEY) \bmod q$** .” (DSS, Ex. 1004, p.14, App. §3.2 (emphases added).) Under the combination of DSS and Rose, therefore, after the hash of KKEY (*i.e.*,  $G(t, KKEY)$ ) is accepted, it is assigned as “**key  $k$** ” and used in performing a cryptographic function. As explained in the preamble, “key  $k$ ” is used in the formula for computing the two digital signature components  **$r$**  and  **$s$** , and thus, “key  $k$ ” is “**use[d] in performing said cryptographic function,**” as claimed. (DSS, *e.g.*, Ex. 1004, p.6-7, §5, p.14, App. §3.2.)

163. Finally, as I explained above, the code in Rose and my proposed pseudocode adaptation based on Rose and DSS performs a modulo operation before returning the resulting value. This would not impact the ultimate value of  $G(t, KKEY)$  because, as explained above, acceptance would only occur if  $G(t, KKEY)$  is less than  **$q$** , in which case “ $G(t, KKEY) \bmod q$ ” would simply result in  $G(t, KKEY)$ . The value provided as “**key  $k$** ” in claim 1[g] is thus the same value that was accepted in claim 1[d] above – it is not affected by the modulo operation.

**2. Dependent Claim 2: The method of claim 1 wherein another seed value is generated by said random number generator if said output is rejected.**

164. This claim adds nothing that was not already addressed in the discussion of claim 1 above. In fact, it is not clear if this claim adds any distinct claim requirements. Claim 1[e] recites that “**if said output  $H(SV)$  is rejected, repeating said method,**” and the first step of the method of claim 1 is “**generating a seed value  $SV$  from a random number generator.**” Nevertheless, as I explained in detail above, under the combination of DSS and Rose, if “**output  $H(SV)$** ” is rejected (*i.e.*, because  $G(t, KKEY)$  in DSS is greater than or equal to  $q$ ), another random number is generated for  $KKEY$  (“seed value”) by the underlying random number generator, and the process continues to claim 1[b], thus repeating the steps of the claim.

**3. Dependent Claim 5: The method of claim 1 wherein said order  $q$  is a prime number represented by a bit string of predetermined length  $L$ .**

165. This claim adds nothing of patentable significance. The specification of the '961 patent admits that the DSA/DSS specifies “a prime number  $q$  of a predetermined bit length, by way of example 160 bits....” ('961, 1:57-62.) DSS similarly states that “[t]he DSA makes use of the following parameters...  $q$  = a prime

divisor of  $p-1$ , where  $2^{159} < q < 2^{160}$ ” (DSS, Ex. 1004, p.6, §4; *see also id.*, p.10, App. §2.1). A person of ordinary skill in the art would have understood that an integer value that falls in this range is a 160-bit value.

166. Indeed, DSS explains that, in the context of generating random numbers for ephemeral key  $k$ , that “[t]hese randomly or pseudorandomly generated integers are selected to be between 0 and the 160-bit prime  $q$  (as specified in the standard).” (DSS, Ex. 1004, p.13, App. §3 (underlining added).) DSS thus discloses that “**said order  $q$  is a prime number represented by a bit string of predetermined length  $L$ ,**” with “ $L$ ” being the length of 160 bits as disclosed in the DSS.

#### **4. Independent Claim 15:**

167. Claim 15 is substantially similar to claim 1, with the exception that claim 15 recites a “computer readable medium” whereas claim 1 recites a method. I will refer back to the discussion above for claim 1 as appropriate, while separately addressing any material differences in claim 15.

- a. “A computer readable medium comprising computer executable instructions for generating a key  $k$  for use in a cryptographic function performed over a group of order  $q$ , said instructions including instructions for:” (Claim 15[Pre])**

168. Claim 15[Pre] of claim 15 is identical to 1[Pre] of claim 1 except that

it recites “a computer readable medium comprising computer executable instructions” that performs the method steps of claim 1.

169. The recitation of “[a] computer readable medium comprising computer executable instructions” adds nothing of patentable significance. DSS explains that “[t]he DSA may be implemented in software, firmware, hardware, or any combination thereof.” (DSS, Ex. 1004, p.3 (underlining added).) It would have been obvious to a person of ordinary skill in the art that an implementation of the DSA “in software” would include computer executable instructions stored on some type of computer readable medium (such as volatile or non-volatile memory). It would also have been apparent and obvious to a person of ordinary skill in the art that the proposed combination of DSS, Schneier and Rose, described for claim 1, could have been implemented in software containing computer executable instructions provided on a computer readable medium. The ’961 patent, in fact, admits that implementing cryptographic functions by a processor executing instructions was well-known. (’961, 3:41-44 (“It will be appreciated that each of these functions is controlled by a processor executing instructions to provide functionality and inter-operability as is well known in the art.”).)

- b. “generating a seed value SV from a random number generator” (Claim 15[a])
- c. “performing a hash function  $H()$  on said seed value SV to provide an output  $H(SV)$ ” (Claim 15[b])
- d. “determining whether said output  $H(SV)$  is less than said order  $q$  prior to reducing mod  $q$ ” (Claim 15[c])
- e. “accepting said output  $H(SV)$  for use as said key  $k$  if the value of said output  $H(SV)$  is less than said order  $q$ ” (Claim 15[d])
- f. “rejecting said output  $H(SV)$  as said key if said value is not less than said order  $q$ ” (Claim 15[e])
- g. “if said output  $H(SV)$  is rejected, repeating said method” (Claim 15[f])
- h. “if said output  $H(SV)$  is accepted, providing said key  $k$  for use in performing said cryptographic function, wherein said key  $k$  is equal to said output  $H(SV)$ ” (Claim 15[g])

170. These seven limitations are identical to limitations 1[a] through 1[g], discussed in detail above, and are therefore rendered obvious for the same reasons.

A side-by-side chart showing these limitations alongside those of claim 1 is below:

Claim 1	Claim 15
a. “generating a seed value SV from a random number generator”	a. “generating a seed value SV from a random number generator”

b. “performing a hash function $H(\ )$ on said seed value SV to provide an output $H(SV)$ ”	b. “performing a hash function $H(\ )$ on said seed value SV to provide an output $H(SV)$ ”
c. “determining whether said output $H(SV)$ is less than said order q prior to reducing mod q”	c. “determining whether said output $H(SV)$ is less than said order q prior to reducing mod q”
d. “accepting said output $H(SV)$ for use as said key k if the value of said output $H(SV)$ is less than said order q”	d. “accepting said output $H(SV)$ for use as said key k if the value of said output $H(SV)$ is less than said order q”
e. “rejecting said output $H(SV)$ as said key if said value is not less than said order q”	e. “rejecting said output $H(SV)$ as said key if said value is not less than said order q”
f. “if said output $H(SV)$ is rejected, repeating said method”	f. “if said output $H(SV)$ is rejected, repeating said method”
g. “if said output $H(SV)$ is accepted, providing said key k for use in performing said cryptographic function, wherein said key k is equal to said output $H(SV)$ ”	g. “if said output $H(SV)$ is accepted, providing said key k for use in performing said cryptographic function, wherein said key k is equal to said output $H(SV)$ ”

171. For the reasons stated for limitations 1[a] through 1[g] of claim 1, claim 15 would have been obvious over DSS in view of Schneier and Rose.

**5. Dependent Claim 16: The computer readable medium of claim 15 wherein another seed value is generated by said random number generator if said output is rejected.**

172. This claim is substantially similar to claim 2, discussed in detail above.



The only difference from claim 2 is “the computer readable medium,” which, as discussed in detail above for the preamble of claim 15, was obvious in light of DSS.

**6. Dependent Claim 19: The computer readable medium of claim 15 wherein said order  $q$  is a prime number represented by a bit string of predetermined length  $L$ .**

173. This claim is substantially similar to claim 5, discussed in detail above. The only difference from claim 5 is “the computer readable medium,” which, as discussed in detail above, was obvious in light of DSS.

**D. Ground 2: Comparison of the Prior Art to the Challenged Claims Based on DSS, Schneier, and Menezes**

174. **Ground 2** is similar to Ground 1 except that, instead of Rose, I have relied on the random number generation technique of Menezes with respect to limitations reciting “determining whether said output  $H(SV)$  is less than said order  $q$  prior to reducing mod  $q$ ,” and the subsequent limitations relating to accepting or rejecting said output  $H(SV)$ . With respect to all other claim limitations, the mapping from **Ground 1** remains the same, as shown below. With respect to dependent claims 2, 5, 16, and 19, the analysis from **Ground 1** remains unchanged and applies here.

**2. Independent Claim 1**

- a. “A method of generating a key  $k$  for use in a cryptographic function performed over a group of order  $q$ , said method including the steps of:” (Claim 1[Pre])**
- b. “generating a seed value  $SV$  from a random number generator” (Claim 1[a])**
- c. “performing a hash function  $H()$  on said seed value  $SV$  to provide an output  $H(SV)$ ” (Claim 1[b])**

175. With respect to the preamble 1[Pre] and claims 1[a]-1[b], the mapping from **Ground 1** remains unchanged. For the reasons stated for Ground 1, the preamble (to the extent limiting) and claims 1[a]-[b] are obvious over the prior art admissions in the '961 patent, and over DSS in view of Schneier.

- d. “determining whether said output  $H(SV)$  is less than said order  $q$  prior to reducing mod  $q$ , accepting said output  $H(SV)$  for use as said key  $k$  if the value of said output  $H(SV)$  is less than said order  $q$ , rejecting said output  $H(SV)$  as said key if said value is not less than said order  $q$ , if said output  $H(SV)$  is rejected, repeating said method” (Claim 1[c]-1[f])**

176. As explained for Ground 1, DSS does not disclose this claim limitation. As mentioned above, **Step 3(a)** of the algorithm for computing the value of  $k$  in DSS states: “ $k = G(t, KKEY) \bmod q$ ” (DSS, Ex. 1004, p.14, App. §3.2 (underlining added)). DSS thus performs a “mod  $q$ ” operation on the result of the hash (“output

H(SV)”), without determining whether or not H(SV) is less than order  $q$ .

177. These limitations of claim 1 are obvious over DSS in view of Menezes. Because claim limitations 1[c] through 1[f] are very closely intertwined and disclosed by overlapping disclosures in the DSS and Menezes, they should be addressed together to avoid needless repetition.

178. Menezes is a book entitled *Handbook of Applied Cryptography* (1997), whose authors include two of the '961 patent named inventors. As I noted in Part VI.A.3, I found nothing in the prosecution history to suggest that Menezes was known or considered by the Patent Office during the prosecution of the '961 patent.

179. Menezes explains that that “[a] *random bit generator* is a device or algorithm which outputs a sequence of statistically independent and unbiased binary digits.” (Menezes, Ex. 1005, p.170, §5.1 (italics in original).) Menezes describes how to employ a random bit generator to generate a random number in a desired range between 0 and  $n$ :

A random bit generator can be used to generate (uniformly distributed) random numbers. For example, a random integer in the interval  $[0, n]$  can be obtained by generating a random bit sequence of length  $\lfloor \lg n \rfloor + 1$ , and converting it to an integer; if the resulting integer exceeds  $n$ , one option is to discard it and generate a new random bit sequence.

(*Id.*, §5.2 (italics in original; underlining added).) Menezes thus discloses a technique for generating uniformly distributed random numbers ranging from 0 to  $n$  by (a) generating a random number in a larger range, and (b) rejecting the number if it lies outside the desired range, *i.e.*, if it exceeds  $n$ . This technique does not use any modulo arithmetic at all, thus avoiding the modulo bias problem discussed above.

180. Menezes, in combination with DSS, discloses “**determining whether said output  $H(SV)$  is less than said order  $q$  prior to reducing mod  $q$** ” as claimed. The “resulting integer” in the passage of Menezes quoted above corresponds to “**said output  $H(SV)$ ,**” and the interval from 0 to  $n$  corresponds to the interval from 0 to  $q-1$  specified by the “order  $q$ .” It would have been obvious to a person of ordinary skill in the art that implementing the simplified rejection technique in Menezes would have required that the software determine whether the random number is less than or equal to  $n$ , or greater than  $n$ , and make a decision based on that determination. The above-quoted passage explains that “if the resulting integer exceeds  $n$ , one option is to discard it and generate a new random bit sequence,” which could not

occur without a comparison of the resulting integer against the value of  $n$ .<sup>24</sup> It therefore would have been obvious, based on the combination of DSS and Menezes, to perform the step of determining whether  $G(t, KKEY)$  (“**said output  $H(SV)$** ”) is less than  $q$ , as required by the claim.

181. With respect to the requirement of determining whether said output  $H(SV)$  is less than said order  $q$  “**prior to reducing mod  $q$ ,**” this is satisfied because

---

<sup>24</sup> The quoted passage of Menezes states that “a random integer in the interval  $[0, n]$  can be obtained by generating a random bit sequence of length  $\lfloor \lg n \rfloor + 1$ .” This means that in order to generate random integers in the range from 0 to  $n$ , one needs to generate a bit sequence containing the number of bits defined by taking the integer portion of  $\log_2(n)$  and adding one. For example, to generate random numbers in the range 0 to 7, one needs to generate  $\lfloor \log_2(7) \rfloor + 1 = 2 + 1 = 3$  bits. This matches what is done in the context of DSS, since  $\lfloor \log_2(q-1) \rfloor + 1 = 160$ , and the number generated by the hash function,  $G(t, KKEY)$ , is 160 bits long.

the technique above avoids the need to perform any “mod  $q$ ” operation altogether. There is no need to perform such an operation because, under the technique in Menezes as combined with DSS, the value of “ $G(t, KKEY)$ ” (“**said output  $H(SV)$** ”) will be rejected unless  $G(t, KKEY) < q$ , in which case  $G(t, KKEY) \bmod q$  would simply return the same value,  $G(t, KKEY)$ .<sup>25</sup>

182. Moreover, it would have been obvious that even in an implementation of the DSS that did perform a “mod  $q$ ” operation on the value “ $G(t, KKEY)$ ,” that the operation would be performed after it was determined whether  $G(t, KKEY) < q$ . This is because Menezes explains that one option of handling an out-of-range random number is to “discard it and generate a new random bit sequence.”

---

<sup>25</sup> In the district court litigation, the Patent Owner has taken the position that “prior to reducing mod  $q$ ” does not actually require performance of a “mod  $q$ ” operation – it instead specifies that if a “mod  $q$ ” operation occurs, it must not take place *before* the determining step has completed. (Rubin Decl., Ex. 1012, ¶57.) I agree with this position.

(Menezes, Ex. 1005, p.170, §5.2.) Under this scenario, it would have been obvious that any “mod  $q$ ” operation could not take place until an acceptable value was attained – which would be after the claimed “determining” step completes. It therefore would have been obvious that the “determining” step would occur “prior to reducing mod  $q$ ,” as claimed.

183. The above-quoted text from Menezes, under the combination with DSS, also discloses “**accepting said output  $H(SV)$  for use as said key  $k$  if the value of said output  $H(SV)$  is less than said order  $q$ .**” This is because Menezes states that “if the resulting integer exceeds  $n$ , one option is to discard it and generate a new random bit sequence.” It would be obvious to a person of ordinary skill in the art that a value that does not exceed  $n$  would be accepted under the teachings of Menezes.

184. And with respect to the step of “**rejecting said output  $H(SV)$  as said key if said value is not less than said order  $q$ , if said output  $H(SV)$  is rejected, repeating said method,**” this occurs under the combination of DSS and Menezes because if the resulting random integer is not less than or equal to  $n$ , “one option is to discard it and generate a new random bit sequence.” (Menezes, Ex. 1005, p.150, §5.2 (underlining added).)

185. ***Manner, Rationale and Motivation to Combine:*** It would have been obvious to combine the techniques of DSS and Schneier with Menezes. This combination would have predictably resulted in the technique of generating **k** in DSS with the following steps:

- (1) Generate a (real or pseudo) random number for KKEY according to DSS alone or in combination with Schneier (the claimed “**seed value SV**”), as explained for claim 1[a];
- (2) compute a hash of KKEY,  $G(t, KKEY)$ , according to DSS (the claimed “**output H(SV)**”), as explained for claim 1[b];
- (3) determine whether  $G(t, KKEY)$  (“**output H(SV)**”) is less than **q**, according to the teachings of Menezes;
- (4) accept  $G(t, KKEY)$  (“**output H(SV)**”) if  $G(t, KKEY)$  **is** less than **q**, according to the teachings of Menezes; and
- (5) reject  $G(t, KKEY)$  if it is **not** less than **q**, by repeating the method and going back to step (1), according to the teachings of Menezes.

186. A person of ordinary skill in the art would have found this to be a trivially simply combination for many of the same reasons applicable to Rose. At the outset, DSS, Schneier, and Menezes are analogous references in the field of random number generation and cryptography, and Menezes contains disclosures reasonably pertinent to the problems to be solved in DSS – generating a random



number for a cryptographic key that falls within a desired range.

187. A person of ordinary skill in the art would have understood that the rejection technique in Menezes could have been readily adapted to any system in which random number generation was needed, and thus could readily have been adapted to DSS to ensure that possible values for “**k**” fall within the range required by the DSS. (DSS, Ex. 1004, p.6, §4, ¶6 (“**k** = a randomly or pseudorandomly generated integer with  $0 < k < q$ ”).) As explained above, the ability to generate random numbers is an integral part of the DSA/DSS. (*Id.*, p.13, Appendix 3 (“Any implementation of the DSA requires the ability to generate random or pseudorandom integers.”).) A way of generating random numbers within the desired range would therefore have been pertinent to a person of ordinary skill in the art implementing the DSA as described in the DSS.

188. I note that Menezes does not specifically describe the potential modulo bias that can occur from performing a modulo operation on a random number. As I have explained throughout this Declaration (and further discussed below), though, modulo bias was a known issue prior to the filing of the ’961 patent. But even if one assumes that a person of ordinary skill in the art would not be aware of this issue, a person of ordinary skill in the art would still be motivated to use the technique in

Menezes in the context of DSS because it solves a particular problem in implementing DSS – ensuring that the resulting value falls within the desired range.

189. Indeed, a person of ordinary skill in the art would have understood that there were a finite number of known and predictable solutions to the problem of obtaining a random or pseudorandom number within a desired range using an underlying generator that outputs a number in a larger range: (1) perform a “modulo operation” (*i.e.*, modular reduction) on the number from the underlying generator, or (2) reject the number from the underlying generator if it falls outside the desired range, and try again (as disclosed in Menezes). Menezes itself acknowledges that when the generated random number exceeds the desired range, “one option is to discard it and generate a new random bit sequence.” (Menezes, Ex. 1005, p.160, §5.2 (underlining added).) Both the Menezes approach and the technique based on modular reduction were known, predictable solutions for generating random numbers within a desired numerical range, and both would have provided a reasonable expectation of success. Thus, the technique in Menezes would have been obvious to try.

190. A person of ordinary skill in the art would also have been motivated to use the technique in Menezes because Menezes expressly states that its technique

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

generates “uniformly distributed” random numbers. (*See* Menezes, Ex. 1005, p.170, §5.2 (“A random bit generator can be used to generate (uniformly distributed) random numbers. For example, a random integer in the interval  $[0, n]$  can be obtained... if the resulting integer exceeds  $n$ , one option is to discard it and generate a new random bit sequence.) (italics in original; underlining added).) It was well-known to persons of ordinary skill in the art that uniformly distributed random numbers (*i.e.*, in which every number within the desired range is equally likely) are preferable for generating cryptographic keys. (Schneier, Ex. 1008, pp.421-22, §17.14; *see also id.*, p.173 (“Good keys are random-bit strings... If the key is 64 bits long, every possible 64-bit key must be equally likely.”) (underlining added).) A person of ordinary skill in the art would thus have been motivated to adapt the technique in Menezes to the DSS to generate **k**.

191. And as explained in the discussion of Rose, *supra*, a person of ordinary skill in the art would have understood that modulo bias could, to some extent, weaken the strength of a cryptographic key by making some numbers more likely to occur than others. The '961 patent admits that it was known that the “mod  $q$ ” operation in the DSA/DSS could create a bias towards lower values in the range of **q**. ('961, 2:53-55 (“With this algorithm, it is to be expected that more values will lie

in the first interval than the second and therefore there is a potential bias in the selection of  $k$ .”.) In fact, the ’961 patent credits third party Daniel Bleichenbacher—not the inventors—with determining a potential impact of the bias in the key  $k$  on the strength of the DSA/DSS. (’961, 2:56-61.) Accordingly, as of the filing of the application for the ’961 patent, the modulo bias problem, and the motivation to avoid it, was known. A person of ordinary skill in the art would thus have been motivated to use the technique in Menezes to maintain “uniformly distributed” random numbers (Menezes, Ex. 1005, p.170, §5.2), resulting in a cryptographically stronger key  $k$ , which would in turn have increased the strength of the resulting digital signatures generated in accordance with the DSS.

192. And as explained in **Ground 1**, a person of ordinary skill in the art would have perceived no technological obstacle to this combination, and would have had every expectation of success. As explained in **Part IV.C** above, the inventors on the ’961 patent were not the first to propose a rejection technique for generating random numbers within a desired range. As explained for **Ground 1**, at least three other programmers on the Internet – Rose, Brecher and Williams – independently came up with substantially the same acceptance-rejection technique described in Menezes, and were able to implement it in a few lines of code. And for the same

reasons explained for **Ground 1**, adapting the technique in Menezes to the generation of the cryptographic key **k** in DSS would have had no significant impact on the performance of the system.

193. In fact, the technique in Menezes could be implemented in a simpler manner than the techniques in Rose. As explained in **Ground 1**, Rose calculates a value (“**biggest**”) based on the largest multiple of the size of the range (“**n**”) less than or equal to the largest random number (“**BIG**”) output by the underlying random number generator. Rose performs this step as an optimization to reduce the likelihood of having to reject random numbers output by the underlying generator. As noted previously, the parameters used by DSS guarantee that the output of the hashing function  $G(t, KKEY)$  (the claimed “output  $H(SV)$ ”) can never equal or exceed twice **q**. As such, the value of “**biggest**” in Rose would always be equal to **q**, and the optimization would never be used. Thus, although Menezes does not

disclose this additional optimization, it would be irrelevant in the context of DSS.<sup>26</sup>

And as with Rose, it is unlikely that the algorithm for generating **k**, adapted according to the teachings of Menezes, would ever have to generate a new random number more than a few times before obtaining an acceptable value less than **q**.

- e. **“if said output H(SV) is accepted, providing said key k for use in performing said cryptographic function, wherein said key k is equal to said output H(SV).” (Claim 1[g])**

194. This step does not add anything that was not already addressed in the discussion of limitations 1[c] through 1[f] above. As explained, DSS and Schneier, in combination with Menezes, disclose a key generation system that accepts the hash of KKEY (“**said output H(SV)**”) if it is less than order **q**.

---

<sup>26</sup> In the context of the rejection method as described in Menezes, (Menezes, Ex. 1005, p.170, §5.2), this optimization is also irrelevant, because the largest value that can be output by the random number generator is never twice or more the upper bound  $n+1$  on the desired range due to the length of the bit sequence generated.

195. Appendix 3 of the DSS identifies **Step 3(a)** of the algorithm for computing the value of **k** as: “ **$k = G(t, KKEY) \bmod q$** .” (DSS, Ex. 1004, p.14, App. §3.2 (emphasis added).) Under the combination of DSS and Menezes, therefore, after the hash of KKEY (*i.e.*  $G(t, KKEY)$ ) is accepted, it is assigned as “**key k**” and used in a cryptographic function. The “mod q” operation specified in Step 3(a) of DSS would not be performed for the reasons explained above. Finally, as explained in the discussion of the preamble, “key k” is used in the formula for computing the two signature components **r** and **s**, and thus “key k” is “**use[d] in performing said cryptographic function,**” as claimed. (DSS, *e.g.*, Ex. 1004, p.6, §5.)

**3. Dependent Claim 2: The method of claim 1 wherein another seed value is generated by said random number generator if said output is rejected.**

196. This claim adds nothing that was not already addressed in the discussion of claim 1 above. As explained in detail above, under the combination of DSS and Menezes, if “**output  $H(SV)$** ” is rejected (*i.e.*, because  $G(t, KKEY)$  in DSS is greater than or equal to **q**), another random number is generated for KKEY and the process starts over.

**4. Dependent Claim 5: The method of claim 1 wherein said order  $q$  is a prime number represented by a bit string of predetermined length  $L$ .**

197. This claim presents no differences with respect to Ground 2 in the same way that it presents no differences with respect to Ground 1. As explained, DSS discloses that, in the context of generating random numbers for ephemeral key  $k$ , that “[t]hese randomly or pseudorandomly generated integers are selected to be between 0 and the 160-bit prime  $q$  (as specified in the standard).” (DSS, Ex. 1004, p.13, App. §3 (underlining added).) DSS thus discloses that “**said order  $q$  is a prime number represented by a bit string of predetermined length  $L$ ,**” with “ $L$ ” being a length of 160 bits as disclosed in the DSS.

**5. Independent Claim 15:**

198. Claim 15 presents no significant differences for purposes of **Ground 2**. This claim recites “a computer readable medium comprising computer executable instructions” that performs the method steps of claim 1. The recitation of “[a] computer readable medium comprising computer executable instructions” adds nothing of patentable significance for the same reasons as in Ground 1.

**6. Dependent Claims 16, 19**

199. These claims present no distinct issues not already addressed by my



analysis of claims 2 and 5, respectively, above. For the reasons stated above, these claims would also have been obvious over DSS, Schneier, and Menezes.

**E. Grounds 3-4: Comparison of the Prior Art from Grounds 1-2, in Further View of Rao and Floyd**

200. **Grounds 3-4** address claims 23, 24, and 27, which recite similar limitations as claims 1, 2, and 5. Independent claim 23 adds a cryptographic unit comprising “**an arithmetic processor**” for performing the steps recited in independent claim 1. The prior art references cited for Grounds 1-2 (DSS, Schneier, Rose, Menezes) do not appear to expressly disclose “an arithmetic processor,” as claimed, but such a processor would have been obvious to a person of ordinary skill in the art. **Grounds 3 and 4** thus cite Rao and Floyd for the claimed arithmetic processor, and rely on the prior art mapping from **Grounds 1 and 2**, respectively, for all other claim limitations.

**2. Independent Claim 23:**

- a. “A cryptographic unit configured for generating a key k for use in a cryptographic function performed over a group of order q, said cryptographic unit having access to computer readable instructions and comprises:” (Claim 23[Pre])**

201. The preamble of claim 23 is substantially similar to independent claims 1 and 15, described in connection with Ground 1 above. For purposes of my analysis

of the prior art, the “**cryptographic unit**” takes the form of a computing device (such as a computer) having a processor for executing computer readable instructions (software) for performing the DSA according to the teachings of DSS, adapted in accordance with the teachings of Schneier and Rose (for Ground 3) and Schneier and Menezes (for Ground 4), as described in detail in my discussion of Grounds 1 and 2 above. This mapping of the “cryptographic unit” to a computer is consistent with the ’961 patent specification, which explains that each of the functions of the cryptographic unit (shown as cryptographic function **20** in Figure 1) “is controlled by a processor executing instructions to provide functionality and inter-operability as is well known in the art.” (’961, 3:41-44.)

202. DSS discloses that “[t]he DSA may be implemented in software, firmware, hardware, or any combination thereof.” (DSS, Ex. 1004, p.3 (underlining added).) It would have been obvious to a person of ordinary skill in the art that an implementation of the DSA on a computer would include hardware (such as a processor) and computer executable instructions stored on some type of computer readable medium (such as volatile or non-volatile memory). It would also have been apparent and obvious to a person of ordinary skill in the art that implementing the proposed combination of DSS, Schneier, and Rose (for Ground 3), or DSS, Schneier,

and Menezes (for Ground 4), could have been implemented in software containing computer readable instructions. The '961 patent admits that carrying out cryptographic functions in this manner was well-known to persons of ordinary skill in the art. ('961, 3:41-44 ("It will be appreciated that each of these functions is controlled by a processor executing instructions to provide functionality and interoperability as is well known in the art.").)

203. The remaining language of the preamble ("generating a key  $k$  for use in a cryptographic function performed over a group of order  $q$ ") is identical to that of claims 1 and 15. For the same reasons stated for Grounds 1 and 2, therefore, these portions of the preamble (to the extent limiting) would have been obvious.

**b. "an arithmetic processor for:" (Claim 23[a])**

204. The specification of the '961 patent devotes only one sentence to the **"arithmetic processor,"** and Figure 1 depicts it as a nondescript black box. (*See* '961, 3:33-38 ("Each of the correspondents **12**, **14** includes a secure cryptographic function **20** including a secure memory **22**, an arithmetic processor **24** for performing finite field operations, a random number generator **26** and a cryptographic hash function **28** for performing a secure cryptographic hash such as SHA-1.") (underlining added), Fig. 1 (item 24).) The phrase "performing finite

field operations” refers to performing arithmetic operations such as addition, subtraction, multiplication, and division, within a range of values, as well as performing comparisons (e.g., determining whether one integer is less than another) and bitwise logical operations (e.g., computing the exclusive OR of two bit strings).

205. The claimed “**arithmetic processor**” thus appears to merely require a processing unit capable of carrying out these types of basic operations. It therefore would have been obvious to a person of ordinary skill in the art – without even having to refer to any additional references – that the “cryptographic unit” used to carry out the combination of DSS, Schneier, and Rose (Ground 3), or DSS, Schneier, and Menezes (Ground 4), would have contained an “arithmetic processor” in order to carry out the functions described by those combinations. For example, Ground 1 presents a combination with Rose, a USENET article that includes source code that utilizes a number of basic mathematical operations including multiplication, division, division with remainder, and comparison. Additionally, it would have been obvious that the SHA hashing function specified in the DSS could have been computed using the mathematical operations described above, and thus, would have been obvious to compute using an arithmetic processor.

206. To the extent there is any question, I have cited Rao and Floyd, which

provide the information about computer architecture and organization that would have formed the basic background knowledge of a person of ordinary skill in the art.

**Rao** is a 1974 textbook that explains how digital computers are divided into functional units or subsystems, including an “arithmetic processor” (AP). (Rao, Ex. 1018, p.39.) Rao explains that “[a]n arithmetic processor is the part of a computer that provides the logic circuits required to perform arithmetic operations such as ADD, SUBTRACT, MULTIPLY, DIVIDE, SQUARE-ROOT, etc.” (*Id.*, p.41 (underlining added).) **Floyd** explains that in a more modern computer that includes an integrated microprocessor, the arithmetic processor functionality is contained within a portion of a computer’s central processing unit (CPU) known as an **arithmetic/logic unit (ALU)**. (Floyd, Ex. 1019, pp.45-46.) “The arithmetic/logic unit (ALU), as the name implies, contains all of the circuitry necessary to perform the computer’s arithmetic and comparison operations.” (*Id.*) Floyd also explains that conventional microprocessors (such as Intel microprocessors) used on personal computers (PCs) included an ALU. (*Id.*, pp.107, p.108 (Fig. 6.2) (“A microprocessor chip contains the ALU as well as the control unit for a microcomputer (photo courtesy of Intel Corporation).”).)

207. It therefore would have been obvious to carry out the DSA technique

described in the DSS (as described in the combinations of **Ground 1** and **Ground 2**) using an “**arithmetic processor**,” as claimed, such as an off-the-shelf, general purpose Intel microprocessor available in December 2000 that carries out arithmetic and logical operations.<sup>27</sup> Indeed, it was common knowledge to persons of ordinary skill in the art that generic, off-the-shelf microprocessors (such as the Intel processors incorporated into countless millions of desktop, laptop, and server computers by December 2000) would have included an ALU for carrying out mathematical and logical operations. It therefore would have been obvious that implementing the DSA technique described in DSS, in the manner described in Ground 1 and Ground 2, would have resulted in the performance of its steps by an arithmetic processor set forth in the remainder of claim 23.

---

<sup>27</sup> In a modern microprocessor, the ALU functionality is physically and functionally integrated with other functions of the processor. I am therefore not mapping the claimed “arithmetic processor” to just the ALU portion of a processor, but rather, to the microprocessor itself that includes the ALU functionality.

208. As noted, a person of ordinary skill in the art would have found this obvious without even having to consult Floyd and Rao. Nevertheless, Floyd and Rao are analogous references in that they provide information reasonably pertinent to the problem of how to implement the DSA in hardware and/or software. Floyd and Rao provide background information about computer architecture that would have been basic knowledge to any person of ordinary skill in the art. Because the claimed “arithmetic processor” was a built-in feature of substantially all computer processors as of December 2000, requiring the steps of limitations 23[a] through 23[g] to be performed via an arithmetic processor would have been obvious.

- c. “generating a seed value SV from a random number generator” (Claim 23[b])
- d. “performing a hash function H( ) on said seed value SV to provide an output H(SV)” (Claim 23[c])
- e. “determining whether said output H(SV) is less than said order q prior to reducing mod q” (Claim 23[d])
- f. “accepting said output H(SV) for use as said key k if the value of said output H(SV) is less than said order q” (Claim 23[e])
- g. “rejecting said output H(SV) as said key if said value is not less than said order q” (Claim 23[f])
- h. “if said output H(SV) is rejected, repeating said method” (Claim 23[g])
- i. “if said output H(SV) is accepted, providing said key k for use in performing said cryptographic function, wherein said key k is equal to said output H(SV)” (Claim 23[h])

209. These seven limitations are identical to limitations 1[a] through 1[g], discussed in detail above. This is confirmed by the following chart:

Claim 1	Claim 23
a. “generating a seed value SV from a random number generator”	b. “generating a seed value SV from a random number generator”
b. “performing a hash function H( ) on said seed value SV to provide an output H(SV)”	c. “performing a hash function H( ) on said seed value SV to provide an output H(SV)”



Claim 1	Claim 23
c. “determining whether said output H(SV) is less than said order q prior to reducing mod q”	d. “determining whether said output H(SV) is less than said order q prior to reducing mod q”
d. “accepting said output H(SV) for use as said key k if the value of said output H(SV) is less than said order q”	e. “accepting said output H(SV) for use as said key k if the value of said output H(SV) is less than said order q”
e. “rejecting said output H(SV) as said key if said value is not less than said order q”	f. “rejecting said output H(SV) as said key if said value is not less than said order q”
f. “if said output H(SV) is rejected, repeating said method”	g. “if said output H(SV) is rejected, repeating said method”
g. “if said output H(SV) is accepted, providing said key k for use in performing said cryptographic function, wherein said key k is equal to said output H(SV)”	h. “if said output H(SV) is accepted, providing said key k for use in performing said cryptographic function, wherein said key k is equal to said output H(SV)”

210. For the reasons stated above, these limitations are disclosed and rendered obvious by DSS, Schneier, and Rose (for Ground 1), and by DSS, Schneier, and Menezes (for Ground 2).

**3. Dependent Claim 24: The cryptographic unit of claim 23 wherein another seed value is generated by said random number generator if said output is rejected.**

211. This claim is substantially similar to claim 2, discussed in detail above for Grounds 1 and 2. The only difference from claim 2 is “the cryptographic unit,”

which, as discussed in detail above for claim 23[a], adds nothing of significance.

**4. Dependent Claim 27: The cryptographic unit of claim 23 wherein said order  $q$  is a prime number represented by a bit string of predetermined length  $L$ .**

212. This claim is substantially similar to claim 5, discussed in detail above for Grounds 1 and 2. The only difference from claim 5 is “the cryptographic unit,” which, as discussed in detail above for claim 23[a], adds nothing of significance.

**VII. CONCLUSION**

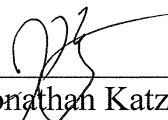
213. In signing this Declaration, I recognize that the Declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross-examination in the case. If required, I will appear for cross-examination at the appropriate time.

214. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. §1001.

Declaration of Jonathan Katz, Ph.D. in Support of  
Petition for *Inter Partes* Review of  
U.S. Patent No. 7,372,961

Dated: 3/28/19

Respectfully submitted,

  
\_\_\_\_\_  
Jonathan Katz, PhD  
Silver Spring, MD

# EXHIBIT A

**Jonathan Katz**  
Department of Computer Science and UMIACS  
University of Maryland  
jkatz@cs.umd.edu

## Education

Ph.D. (with distinction), Computer Science, Columbia University, 2002

**Dissertation:** *Efficient Cryptographic Protocols Preventing “Man-in-the-Middle” Attacks*

**Advisors:** Zvi Galil and Moti Yung

Also advised by Rafail Ostrovsky (Telcordia Technologies)

M.Phil., Computer Science, Columbia University, 2001

M.A., Chemistry, Columbia University, 1998

S.B., Mathematics, Massachusetts Institute of Technology, 1996

S.B., Chemistry, Massachusetts Institute of Technology, 1996

## Employment History

**Director**, Maryland Cybersecurity Center (MC2)

*October, 2013 – present*

**Professor**, University of Maryland

*July, 2013 – present*

**Associate Professor**, University of Maryland

*July, 2008 – June, 2013*

**Assistant Professor**, University of Maryland

*July, 2002 – June, 2008*

Responsible for maintaining a world-class research program in cryptography and information security. Duties include supervising graduate students and designing and teaching courses in cryptography, theoretical computer science, and network security.

**Independent Cryptography/Cybersecurity Consultant**, various positions

*August, 2002 – present*

I have consulted for several companies and government agencies on the design, analysis, and implementation of cryptographic protocols and algorithms. I have also delivered tailored courses on a wide range of topics in cryptography and cybersecurity to audiences in industry, academia, and government. Finally, I have provided expert testimony in intellectual property disputes.

**Visiting Research Scientist**, IBM T.J. Watson Research Center (Hawthorne, NY)

*August, 2008 – July, 2009*

Visited and collaborated with the cryptography research group at IBM.

**Visiting Professor**, École Normale Supérieure (Paris, France)

*June – July, 2008*

Presented three lectures on my research; collaborated with the cryptography research group at ENS.

**Research Fellow**, Institute for Pure and Applied Mathematics, UCLA

*September – December, 2006*

Invited as a core participant for the Fall 2006 program on “Securing Cyberspace: Applications and Foundations of Cryptography and Computer Security.”

**Visiting Research Scientist**, DIMACS

*March – May, 2002*

Conducted research in both theoretical and applied cryptography, leading to two published papers.

**Instructor**, Columbia University

*Summer, 1999 – Spring, 2002*

Taught *Computability and Models of Computation* (Summer '01, Spring '02), *Introduction to Cryptography* (Spring '01), and *Introduction to Computer Programming in C* (Summer '99, Spring '00).

**Research Scientist**, Telcordia Technologies

*March, 2000 – October, 2001*

Member of the Mathematical Sciences Research Center. Conducted basic research in cryptography leading to the filing of two provisional patents. Provided security consulting services for other research groups within Telcordia.

**Security Consultant**, Counterpane Systems

*May, 1999 – March, 2000*

Discovered security flaws in email encryption software (PGP); this work was widely covered in the press and led to two published papers and a refinement of the current standards for email encryption. Designed and implemented secure web-based protocols for clients. Contributed to *Secrets and Lies: Digital Security in a Networked World*, by B. Schneier (J. Wiley & Sons, 2000).

## Honors and Awards

University of Maryland Distinguished Scholar-Teacher Award, 2017–2018

Member, State of Maryland Cybersecurity Council and Chair, Subcommittee on Education and Workforce Development, 2015–present

Member, steering committee, IEEE Cybersecurity Initiative, 2014–2017

Humboldt Research Award, 2015

Named one of Daily Record’s “50 Influential Marylanders,” 2014

Invited participant, DARPA Computer Science Study Group, 2009–2010

NSF CAREER award, 2005–2010

University of Maryland GRB semester award, 2005–2006

National Defense Science and Engineering Graduate Fellowship, 1996–1999

NSF Graduate Fellowship, 1996 (declined)

Alpha Chi Sigma award for academic excellence, MIT, 1996

## Research Grants

(Dollar amounts listed reflect the University of Maryland portion of the award. Unless indicated otherwise, I am the sole PI from the University of Maryland on the award.)

“Practical Multi-Party Computation for Lightweight Clients,” Alibaba, \$100,000.

*January, 2019 – December, 2019*

“Efficient Implementations of Secure Multi-Party Computation,” PlatON, \$142,214.

*September, 2018 – March, 2020*

“CPS: Medium: Collaborative Research: Security vs. Privacy in Cyber-Physical Systems,” NSF (CNS-1837517), \$360,000.

*September, 2018 – August, 2021*

“Scholarship for Service (SFS) for ACES,” NSF (DGE-1753857), \$5,046,316.

*January, 2018 – December, 2022*

PI: Michel Cukier; co-PIs: Jonathan Katz, Lawrence Gordon, William Nolte, and Jan Plane

“LL/University of Maryland Research Collaboration on Secure Multi-Party Computation,” Lincoln Laboratory, \$49,892.

*April, 2017 – May, 2018*

“Automated Analysis and Synthesis of Secure Cryptographic Algorithms,” NRL, \$266,004.

*September, 2016 – September, 2019*

“TWC: Medium: Collaborative: New Protocols and Systems for RAM-Based Secure Computation,” NSF (CNS-1563722), \$484,196.

*May, 2016 – April, 2019*

PI: Jonathan Katz; co-PI: Mike Hicks

“Design and Analysis of (Quantum-Resistant) Hash-Based Signatures,” Cisco, \$68,694.

*April, 2016 – March, 2017*

“Provable Security for Next-Generation Cryptography,” NIST, \$1,097,937.

*September, 2015 – August, 2018*

PI: Jonathan Katz; co-PIs: Dana Dachman-Soled and Babis Papamanthou

“TWC: Large: Collaborative: The Science and Applications of Crypto-Currency,” NSF (CNS-1518765), \$1,935,783.

*July, 2015 – June, 2018*

PI: Elaine Shi; co-PIs: Michael Hicks, Jonathan Katz, and David Van Horn

“TWC: Medium: Apollo: An Architecture for Scalable Verifiable Computing,” NSF (CNS-1514261), \$1,200,000.  
*July, 2015 – June, 2018*  
 PI: Babis Papamanthou; co-PIs: Amol Deshpande, Jonathan Katz, and Elaine Shi

“US-Europe Workshop on Cryptography and Hardware Security for the Internet of Things,” ARO, \$35,000.  
*June, 2015 – June, 2016*  
 PI: Gang Qu; co-PI: Jonathan Katz

“Secure Information Flows in Hybrid Coalition Networks,” US Army Research Laboratory/UK Ministry of Defence (International Technology Alliance in Network and Information Science), \$179,708.  
*May, 2015 – May, 2016*  
 PI: Michael Hicks; co-PI: Jonathan Katz

“Secure Network-Centric Data Distribution and Processing,” US Army Research Laboratory/UK Ministry of Defence (International Technology Alliance in Network and Information Science), \$64,525.  
*May, 2015 – May, 2016*

“EAGER: Physical, Social, and Situational Factors as Determinants of Public WiFi Users’ Online Behaviors,” NSF (CNS-1444633), \$215,002.  
*October, 2014 – September, 2016*  
 co-PIs: Jonathan Katz and David Maimon

“Establishing a Science of Security Research Lablet at the University of Maryland,” NSA, \$4,737,089.  
*March, 2014 – March, 2017*  
 Lead PI: Jonathan Katz

“Automating Secure Computation,” DARPA (via subcontract to ACS), \$51,213.  
*January, 2014 – February, 2015*  
 PI: Elaine Shi; co-PI: Jonathan Katz

“Network Security: Efficient Protocols for Message Integrity in DTNs,” Laboratory for Telecommunications Sciences, \$176,353.  
*April, 2013 – March, 2015*

“Secure Information Flows in Hybrid Coalition Networks,” US Army Research Laboratory/UK Ministry of Defence (International Technology Alliance in Network and Information Science), \$356,615.  
*May, 2013 – May, 2015*  
 PI: Michael Hicks; co-PI: Jonathan Katz

“Secure Network-Centric Data Distribution and Processing,” US Army Research Laboratory/UK Ministry of Defence (International Technology Alliance in Network and Information Science), \$108,016.  
*May, 2013 – May, 2015*



“TWC: Small: Exploring Cryptographic Models and Setup Assumptions,” NSF (CNS-1223623), \$400,945.

*September, 2012 – August, 2015*

“Developing a Science of Cybersecurity,” US Army Research Laboratory, \$2,813,768.

*October, 2011 – September, 2013*

“TC: Large: Collaborative Research: Practical Secure Two-Party Computation: Techniques, Tools, and Applications,” NSF (CNS-1111599), \$1,000,000.

*August, 2011 – August 2016*

PI: Jonathan Katz; co-PI: Michael Hicks

“Delegated, Outsourced, and Distributed Computation,” US Army Research Laboratory/UK Ministry of Defence (International Technology Alliance in Network and Information Science), \$199,226.

*May, 2011 – April, 2013*

“Toward Practical Cryptographic Protocols for Secure Information Sharing, Phase II CSSG,” DARPA, \$400,000.

*September, 2010 – August, 2012*

“NetSE: Medium: Collaborative Research: Privacy-Preserving Social Systems,” NSF (IIS-0964541), \$880,000.

*September, 2010 – August, 2013*

PI: Bobby Bhattacharjee; co-PIs: Jonathan Katz and Neil Spring

Supplement for “CAREER: Models and Cryptographic Protocols for Unstructured, Decentralized Systems,” NSF (CNS-0447075), \$80,000.

*August, 2009 – August, 2010*

“Energy Efficient Security Architectures and Infrastructures,” US Army Research Laboratory/UK Ministry of Defence (International Technology Alliance in Network and Information Science), \$162,450.

*May, 2009 – April, 2011*

“Cryptographic Primitives and Protocols for Security in Complex Systems,” DARPA, \$100,000.

*March, 2009 – March, 2010*

“Understanding Fairness in Secure Two-Party and Multi-Party Computation,” NSF (CCF-0830464), \$277,782.

*September, 2008 – August, 2011*

“Collaborative Research: CT-ISG: Efficient Cryptography Based on Lattices,” NSF (CNS-0716651), \$138,500.

*September, 2007 – August, 2010*

“Efficient Security Techniques for Information Flows in Coalition Environments,” US Army Research Laboratory/UK Ministry of Defence (International Technology Alliance in Network and Information Science), \$395,026.

*May, 2007 – April, 2009*

PIs: Jonathan Katz and Michael Hicks

“Designing Reliable and Secure Tactical MANETs,” DoD MURI, \$1,442,324.

*May, 2007 – April, 2012*

PI: Virgil Gligor; co-PIs: John Baras and Jonathan Katz

“New Techniques for Authenticating Humans (and Other Resource-Constrained Devices),” NSF (CNS-0627306), \$300,000.

*September, 2006 – August, 2009*

“Feasibility and Efficiency of Secure Computation,” United States-Israel Binational Science Foundation, \$120,000.

*September, 2005 – August, 2009*

“CAREER: Models and Cryptographic Protocols for Unstructured, Decentralized Systems,” NSF (CNS-0447075), \$400,000.

*February, 2005 – January, 2010*

“Secure Design and Usage of Cryptographic Hash Functions,” University of Maryland GRB semester award.

*2005–2006 academic year*

“ITR-(ASE+NHS)-(DMC+INT+SOC): Resilient Storage and Querying in Decentralized Networks,” NSF (CNS-0426683), \$720,000.

*September, 2004 – August, 2008*

PI: Bobby Bhattacharjee; co-PIs: Sudarshan Chawathe, Jonathan Katz, and Aravind Srinivasan

“Distributed Trust Computations for Decentralized Systems,” NSF (CNS-0310499), \$375,000.

*August, 2003 – July, 2006*

PI: Bobby Bhattacharjee; co-PI: Jonathan Katz

“Collaborative Research: Mitigating the Damaging Effects of Key Exposure,” NSF (CNS-0310751), \$240,000.

*August, 2003 – July, 2006*

## PhD Students

### Graduated:

Yupeng Zhang (graduated in 2018, co-advised with Babis Papamanthou)

Will join Texas A&M as an assistant professor after a postdoc at UC Berkeley

Xiao Wang (graduated in 2018)

Will join Northwestern University as an assistant professor after a postdoc at MIT/BU

Kartik Nayak (graduated in 2018, co-advised with Elaine Shi)

Will join Duke University as an assistant professor after a postdoc at VMware

Daniel Apon (graduated in 2017)

Currently researcher at NIST

Aishwarya Thiruvengadam (graduated in 2017, co-advised with Dana Dachman-Soled)

Currently postdoctoral researcher at UCSB

Andrew Miller (graduated in 2016, co-advised with Elaine Shi)  
Currently assistant professor at UIUC

Alex Malozemoff (graduated in 2016)  
Currently at Galois, Inc.

Adam Groce (graduated in 2014)  
Currently assistant professor at Reed College

Ranjit Kumaresan (graduated in 2012)  
Currently at Microsoft Research, Redmond

Arkady Yerukhimovich (graduated in 2011)  
Currently assistant professor at George Washington University

S. Dov Gordon (graduated in 2010)  
Currently assistant professor at George Mason University

Omer Horvitz (graduated in 2007, co-advised with Prof. Gligor)  
Currently at techmeme.com

Chiu-Yuen Koo (graduated in 2007)  
Currently at Google Labs, Mountain View, CA

Ruggero Morselli, (graduated in 2006, co-advised with Prof. Bhattacharjee)  
Currently at Google Labs, Pittsburgh, PA

## Postdoctoral Researchers

Daniel Genkin, 2016 – 2018  
Currently assistant professor at the University of Michigan

Samuel Ranellucci, 2016 – 2018  
Currently at Unbound Tech

Dimitris Papadopoulos, 2016 – 2017  
Currently assistant professor at Hong Kong University

Jacob Alperin-Sheriff, 2015 – 2016  
Currently researcher at NIST

Hoang Viet Tung, 2014 – 2015  
Currently assistant professor at Florida State University

Feng-Hao Liu, 2013 – 2015  
Currently assistant professor at Florida Atlantic University

Jean Paul Degabriele, 2013 – 2014  
Currently postdoctoral researcher at Royal Holloway University of London

Yan Huang, 2012 – 2014  
Currently assistant professor at Indiana University

Hong-Sheng Zhou, 2010 – 2013  
Currently assistant professor at Virginia Commonwealth University

Dominique Schröder, 2011 – 2012

Currently professor and chair for applied cryptography at University of Erlangen-Nuremberg

Raef Bassily, 2012

Current assistant professor at The Ohio State University

Seung Geol Choi, 2010 – 2012

Currently associate professor at the US Naval Academy

Vassilis Zikas, 2010 – 2012

Currently associate professor at University of Edinburgh

Lior Malka, 2009 – 2010

Ik Rae Jeong, 2005 – 2006

Currently professor at Korea University

## Professional Activities

### Editorial board:

- Journal of Cryptology (2011–present)
- International Journal of Applied Cryptography (2007–present)
- Proceedings on Privacy Enhancing Technologies (2015–2017)
- Information & Computation (2012–2017)
- Journal of Computer and System Sciences (2013–2014)
- IET Information Security (2005–2012)
- Fundamenta Informaticae (2006–2011)

### Program chair:

- ACM Conference on Computer and Communications Security 2019–2020
- ACM Conf. on Computer and Communications Security (area chair, cryptography) 2018
- Crypto 2016–2017
- Symposium and Bootcamp on the Science of Security (HoTSoS) 2017
- Intl. Conference on Practice and Theory in Public-Key Cryptography (PKC) 2015
- Conference on Decision and Game Theory for Security (GameSec) 2011
- Cryptography Track, 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS) 2010
- Applied Cryptography and Network Security (ACNS) 2007

### Organizer:

- Winter School on Cryptocurrency and Blockchain Technologies (Shanghai, China), 2017

### Steering committees:

- Co-chair, IEEE Cybersecurity Award Selection Committee (2017, 2018)
- IEEE Cybersecurity Initiative (2014–2017)
- International Symposium on Cyber Security, Cryptography and Machine Learning (CSCML), 2016–present

### Program committees:

- Real World Cryptography (RWC) 2019

- ACM Conf. on Computer and Comm. Security (CCS) 2005, 2006, 2011–2013, 2017, 2018
- Computer Security and Information Privacy Track, 19th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS) 2017
- IndoCrypt 2017
- IEEE European Symposium on Security & Privacy 2016, 2017
- Network and Distributed System Security (NDSS) 2016
- Theory of Cryptography Conference (TCC) 2006, 2007, 2012, 2016
- Mycrypt 2016
- Symposium and Bootcamp on the Science of Security (HotSoS) 2015, 2016, 2018
- IEEE Symposium on Security & Privacy (Oakland) 2009, 2015
- European Symposium on Security in Computer Security (ESORICS) 2013
- Crypto 2003, 2005, 2006, 2009, 2013
- Eurocrypt 2006, 2008, 2009, 2011, 2013
- Asiacrypt 2004, 2007, 2008, 2010, 2012
- RSA—Cryptographers’ Track 2006, 2007, 2010, 2012
- Financial Cryptography 2012
- ACM-SIAM Symposium on Discrete Algorithms (SODA) 2011
- Intl. Conf. on Cryptology and Network Security (CANS) 2010
- Intl. Conf. on Pairing-Based Cryptography (Pairing) 2010
- Public-Key Cryptography (PKC) 2007, 2010
- ACM Symposium on Theory of Computing (STOC) 2009
- Applied Cryptography and Network Security (ACNS) 2006, 2009
- IEEE Symposium on Foundations of Computer Science (FOCS) 2008
- Security in Communication Networks 2008
- ICALP 2007
- ACM Workshop on Security and Sensor Networks (SASN) 2004, 2005, 2006
- Security and Cryptography for Networks (SCN) 2006
- VietCrypt 2006
- International Conference on Information Security and Cryptology (ICISC) 2005, 2006
- UCLA/IPAM workshop on “Locally decodable codes...” 2006
- Workshop on Cryptography over Ad Hoc Networks (WCAN) 2005, 2006
- International Conference on Cryptology in Malaysia (Mycrypt) 2005
- Workshop in Information Security and Applications (WISA) 2004

## Courses/Tutorials

3-hour tutorial: “Introduction to Secure Computation,” 1st Crypto Innovation School (Shenzhen, China), November 2018.

3-hour tutorial: “Incentives and Game-Theoretic Considerations in Bitcoin,” Winter School on Cryptocurrency and Blockchain Technologies (Shanghai, China), January 2017.

7-week on-line course: “Cryptography,” Coursera, 2014.

1-hour tutorial: “Message Authentication Codes (an Introduction),” Army Research Laboratory (Adelphi, MD), October 2009.

Half-day tutorial: “Ruminations on Defining Rational Multi-Party Computation,” Summer School on Rational Cryptography (Bertinoro, Italy), June 2008.

1-hour tutorial: “The Basics of Public-Key Encryption,” Booz Allen Hamilton (Linthicum, MD), October 2007.

2<sup>+</sup>-hour tutorial: “A Survey of Modern Cryptography,” ACM Sigmetrics, June 2007.

Week-long course: “Zero Knowledge: Foundations and Applications,” (Bertinoro, Italy), October 2006.

Half-day tutorial: “Black-Box Reductions, Impossibility Results, and Efficiency Lower Bounds,” UCLA/IPAM, September 2006.

## **Invited Panel and Session Participation**

ATARC Federal CISO Summit: moderator, “Addressing the Cybersecurity Skills Gap,” January 2018.

Big Data in Finance, University of Michigan: panel participant, October 2016.

3rd Annual Conference on Cyber Security and the Law (French American Foundation, Washington, D.C.): panel participant, September 2016.

11th Colloquium for Information System Security Education (Boston University): panel member, “How to Teach Cryptology,” June 2007.

## **Invited Talks**

DC-area Security and Privacy Seminar: “How to Hash: Efficient and Secure Multiparty Computation from Fixed-Key Block Ciphers,” February 2019.

TPMPC Workshop (Aarhus, Denmark): “Optimizing ZK Proofs from Secure Computation,” May 2018.

Sandia National Laboratories: “A Survey of Secure Computation,” March 2018.

UT Dallas Distinguished Lecture Series: “Post-Quantum Signatures from Secure Computation,” November 2017.

New Jersey Institute of Technology Distinguished Speaker Series: “Post-Quantum Signatures from Secure Computation,” October 2017.

Cyberweek—Academic Perspectives on Cybersecurity Challenges (Tel Aviv, Israel): “Secure Distributed Computation,” June 2017.

2nd Hebrew University Networking Summer (Jerusalem, Israel): “Recent Progress in Generic Secure Computation,” June 2017.

MITRE Distinguished Speaker Series (McLean, VA): “Recent Progress in Efficient Secure Computation,” May 2017.

IEEE Cybersecurity Development Conference (Boston, MA): “How to Think about Cryptography: Common Crypto Flaws and How to Avoid Them,” November 2016.

Cyber Community of Interest Meeting, ONR: “Automated Analysis and Synthesis of Symmetric-Key Modes of Encryption,” September 2016.

Computing Research Association: Addressing National Priorities and Societal Needs (Computing Community Consortium, Washington, D.C.): “Better Privacy and Security via Secure Multiparty Computation,” May 2016.

Workshop on Human Factors in Cybersecurity Design, Hebrew University (Jerusalem, Israel): “Taking the Human into Account in Cryptographic Design,” March 2016.

George Washington University Department of Mathematics: “The Nature of Proofs: A Computational Perspective,” February 2016.

Foundations of Cyber Security and Privacy Symposium, Max Planck Society (Munich, Germany): “Cryptography as a Nucleus for Cybersecurity Research,” July 2015.

Privacy Enhancing Technologies Symposium (PETS) 2015: “Secure Computation: Where Do We Go From Here?” June 2015.

Naval Postgraduate School Foundation, President’s Circle Retreat: “Privacy-Preserving Distributed Computation,” April 2014.

Georgetown University: “Secure Computation in the RAM Model,” April 2014.

Rutgers University: “Privacy-Preserving Computation: How, What, and Why?” November 2013.

First EasyCrypt workshop (University of Pennsylvania): “EasyCrypt 0.2 Feedback and Recommendations,” July 2013.

Workshop on Real-World Cryptography (Stanford): “Practical Anonymous Subscriptions,” January 2013.

Workshop on Theory and Practice of Multiparty Computation (Aarhus, Denmark): “Recent Results on Game Theory and Secure Computation,” June 2012.

Indiana University: “Is (Generic) Secure Two-Party Computation Practical?” November 2011.

Microsoft Research (Redmond, WA): “(Ever More) Efficient Secure Two-Party Computation,” March 2011.

PerAda Workshop on Security, Trust, and Privacy (Rome, Italy): “Privacy, Trust, and Security in Pervasive Computing: Challenges and Opportunities,” November 2010.

Tsinghua University (Beijing, China): “Fairness and Partial Fairness in Two-Party Computation,” June 2010.

Beijing Institute of Technology: “Rational Secret Sharing,” June 2010.

SKLOIS: The State Key Laboratory Of Information Security (Beijing, China): “Leakage-Resilient Cryptography,” June 2010.

SKLOIS: The State Key Laboratory Of Information Security (Beijing, China): “Rational Secret Sharing,” June 2010.

Workshop on Decentralized Mechanism Design, Distributed Computing, and Cryptography (Princeton University): “Rational Secret Sharing: A Survey,” June 2010.

Microsoft Research (Cambridge, MA): “Rational Secret Sharing,” April 2009.

AT&T Labs: “Fairness and Partial Fairness in Secure Two-Party Computation,” February 2009.

University of Toronto: “Fairness and Partial Fairness in Secure Two-Party Computation,” February 2009.

Joint Mathematics Meetings, AMS Special Session on Algebraic Cryptography and Generic Complexity: “Public-Key Cryptography from a (Theoretical) Cryptographer’s Perspective,” January 2009.

Dagstuhl workshop on Theoretical Foundations of Practical Information Security (Germany): “Partial Fairness in Secure Two-Party Computation,” December 2008.

École Normale Supérieure (Paris, France): “Efficient Cryptographic Protocols Based on the Hardness of Learning Parity with Noise,” July 2008.

École Normale Supérieure (Paris, France): “Predicate Encryption: A New Paradigm for Public-Key Encryption,” July 2008.

École Normale Supérieure (Paris, France): “Fairness in Secure Computation,” June 2008.

UC Berkeley: “Predicate Encryption: A New Paradigm for Public-Key Encryption,” May 2008.

5th Theory of Cryptography Conference (TCC) 2008 (New York): “Bridging Game Theory and Cryptography: Recent Results and Future Directions,” March 2008.

MIT Cryptography and Information Security Seminar: “Complete Fairness in Secure Two-Party Computation,” March 2008.

11th IMA Intl. Conference on Cryptography and Coding Theory (Cirencester, UK): “Efficient Cryptographic Protocols Based on the Hardness of Learning Parity with Noise,” December 2007.

INDOCRYPT 2007 (Chennai, India): “Capability-Based Encryption: A New Paradigm for Public-Key Encryption,” December 2007.

Pennsylvania State University: “Universally-Composable Multi-Party Computation using Tamper-Proof Hardware,” April 2007.

Workshop on Cryptography: Underlying Mathematics, Provability, and Foundations (Fields Institute, Toronto): “Blind Signatures: Definitions and Constructions,” November 2006.

Workshop on Foundations of Secure Multi-Party Computation (UCLA/IPAM): “On Expected Constant-Round Protocols for Broadcast,” November 2006.

Workshop on Public-Key Systems with Special Properties (UCLA/IPAM): “Blind Signatures: Definitions and Constructions,” October 2006.

13th SIAM Meeting on Discrete Mathematics (Victoria, Canada): “New Techniques for Authenticating Humans,” June 2006.



Boston University: “New Techniques for Authenticating Humans (and other Resource-Constrained Devices),” April 2006.

Stevens Institute of Technology: “New Techniques for Authenticating Humans (and other Resource-Constrained Devices),” March 2006.

Georgia Tech: “New Techniques for Authenticating Humans (and other Resource-Constrained Devices),” November 2005.

University of Modena: “Secure Authentication without Traditional Cryptographic Keys,” July 2005.

Workshop on the Past, Present, and Future of Oblivious Transfer (Haifa, Israel): “Round-Optimal Secure Two-Party Computation,” May, 2005.

UCLA: “Secure Remote Authentication Using Biometric Data,” March, 2005.

Luminy Workshop on Cryptography (Marseilles, France): “Secure Remote Authentication Using Biometric Data,” November, 2004.

DIMACS Workshop on *Cryptography: Theory Meets Practice*: “Using Biometric Data for Secure Network-Based Authentication,” October, 2004.

MIT Cryptography and Information Security Seminar: “Round-Optimal Secure Two-Party Computation,” April, 2004.

Korea University: “Scalable and Efficient Protocols for Authenticated Group Key Exchange,” November, 2003.

Korea Information Security Agency (KISA): “Efficient Protocols for Password-Only Authenticated Key Exchange,” November, 2003.

6th Annual International Conference on Information Security and Cryptology (ICISC 2003): “Binary Tree Encryption: Constructions and Applications,” November, 2003.

National Science Foundation (NSF) — Washington Area Trustworthy Systems Hour: “Maintaining Security in the Event of Key Exposure,” April, 2003.

New York University: “Efficient and Non-Malleable Proofs of Plaintext Knowledge and Applications,” July, 2002.

IBM T.J. Watson Research Center: “A Forward-Secure Public-Key Encryption Scheme,” July, 2002.

DIMACS Workshop on *Cryptographic Protocols in Complex Environments*: “Efficient and Non-Malleable Proofs of Plaintext Knowledge and Applications,” May, 2002.

IBM T.J. Watson Research Center: “Practical Password-Authenticated Key Exchange Provably Secure Against Off-Line Dictionary Attacks,” December, 2000.

MIT Cryptography and Information Security Seminar: “Practical and Provably Secure Password-Authenticated Key Exchange,” December, 2000.

Bell Labs (Lucent Technologies) Crypto/Security Seminar: “Cryptographic Counters and Applications to Electronic Voting,” November, 2000.

## Publications

### Books Authored or Edited

1. J. Katz and H. Shacham, eds. *Advances in Cryptology—Crypto 2017, Proceedings*. LNCS vols. 10401–10403, Springer, 2017.
2. J. Katz and M. Robshaw, eds. *Advances in Cryptology—Crypto 2016, Proceedings*. LNCS vols. 9814–9816, Springer, 2016.
3. J. Katz, ed. *Public-Key Cryptography (PKC) 2015, Proceedings*. LNCS vol. 9020, Springer, 2015.
4. J. Katz and Y. Lindell. *Introduction to Modern Cryptography, second edition*. Chapman & Hall/CRC Press, 2014. (First edition published in 2007.)
5. J.S. Baras, J. Katz, and E. Altman. *Decision and Game Theory for Security, Second Intl. Conference, GameSec 2011, Proceedings*. LNCS vol. 7037, Springer, 2011.
6. J. Katz. *Digital Signatures*. Springer, 2010.
7. J. Katz and M. Yung, eds. *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Proceedings*. LNCS vol. 4521, Springer, 2007.

### Book Chapters

1. J. Katz. “Cryptography.” In *Computing Handbook (3rd edition), vol. 1: Computer Science and Software Engineering*, A. Tucker, T. Gonzalez, and J. Diaz-Herrera, eds., Chapman & Hall/CRC Press, 2014.
2. J. Katz. “Public-Key Cryptography.” In *Handbook of Information and Communication Security*, P. Stavroulakis and M. Stamp, eds., Springer, 2010.
3. J. Katz. “Cryptography.” In *Wiley Encyclopedia of Computer Science and Engineering*, B.W. Wah, ed., John Wiley & Sons, 2008.
4. J. Katz. “Symmetric-Key Encryption.” In *The Handbook of Information Security*, H. Bidgoli, ed., John Wiley & Sons, Inc., 2005.
5. J. Katz. “Cryptography.” In *Computer Science Handbook, 2nd edition*, A. Tucker, ed., CRC Press, 2004.

### Journal Articles

1. T. Chakraborty, S. Jajodia, J. Katz, A. Picariello, G. Sperli, and V.S. Subrahmanian. “FORGE: A Fake Online Repository Generation Engine for Cyber Deception.” *IEEE Trans. Dependable and Secure Computing*, to appear.
2. Y. Zhang, C. Papamanthou, and J. Katz. “Verifiable Graph Processing.” *ACM Trans. on Privacy and Security*, to appear.

3. S.G. Choi, J. Katz, D. Schröder, A. Yerukhimovich, and H.-S. Zhou. “(Efficient) Universally Composable Oblivious Transfer Using a Minimal Number of Stateless Tokens.” *J. Cryptology*, to appear. **One of three papers from TCC 2014 invited to this journal.**
4. M. Lee, A. Dunn, J. Katz, B. Waters, and E. Wichel. “Anon-Pass: Practical Anonymous Subscriptions.” *IEEE Security & Privacy* 12(3): 20–27, 2014. **Invited to a special issue for papers from the IEEE Symp. on Security & Privacy, 2014.**
5. S. D. Gordon, J. Katz, R. Kumaresan, and A. Yerukhimovich. “Authenticated Broadcast with a Partially Compromised Public-Key Infrastructure.” *Information & Computation* 234: 17–25, 2014. **Invited to a special issue of this journal for papers from SSS 2010.**
6. D. Apon, J. Katz, and A. Malozemoff. “One-Round Multi-Party Communication Complexity of Distinguishing Sums.” *Theoretical Computer Science* 501: 101–108, 2013.
7. J. Katz and V. Vaikuntanathan. “Round-Optimal Password-Based Authenticated Key Exchange.” *J. Cryptology* 26(4): 714–743, 2013. **One of three papers from TCC 2011 invited to this journal.**
8. J. Katz, A. Sahai, and B. Waters. “Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products.” *J. Cryptology* 26(2): 191–224, 2013. **One of four papers from Eurocrypt 2008 invited to this journal.**
9. Y. Dodis, B. Kanakurthi, J. Katz, L. Reyzin, and A. Smith. “Robust Fuzzy Extractors and Authenticated Key Agreement from Close Secrets.” *IEEE Transactions on Information Theory* 58(9): 6207–6222, 2012.
10. J. Katz, P. MacKenzie, G. Taban, and V. Gligor. “Two-Server Password-Only Authenticated Key Exchange.” *J. Computer and System Sciences* 78(2): 651–669, 2012.
11. J. Katz. “Which Languages Have 4-Round Zero-Knowledge Proofs?” *J. Cryptology* 25(1): 41–56, 2012. **One of three papers from TCC 2008 invited to this journal.**
12. S.D. Gordon and J. Katz. “Partial Fairness in Secure Two-Party Computation.” *J. Cryptology* 25(1): 14–40, 2012.
13. S.D. Gordon, C. Hazay, J. Katz, and Y. Lindell. “Complete Fairness in Secure Two-Party Computation.” *J. of the ACM* 58(6): 1–36, 2011.
14. Y. Ishai, J. Katz, E. Kushilevitz, Y. Lindell, and E. Petrank. “On Achieving the ‘Best of Both Worlds’ in Secure Multiparty Computation.” *SIAM J. Computing* 40(1): 122–141, 2011.
15. J. Katz, J.-S. Shin, and A. Smith. “Parallel and Concurrent Security of the HB and HB<sup>+</sup> Protocols.” *J. Cryptology* 23(3): 402–421, 2010.

16. O. Horvitz and J. Katz. “Bounds on the Efficiency of ‘Black-Box’ Commitment Schemes.” *Theoretical Computer Science* 411(10): 1251–1260, 2010. **Invited to a special issue of this journal.**
17. J. Katz, R. Ostrovsky, and M. Yung. “Efficient and Secure Authenticated Key Exchange Using Weak Passwords.” *J. of the ACM* 57(1): 78–116, 2009.
18. J. Katz, C.-Y. Koo, and R. Kumaresan. “Improving the Round Complexity of VSS in Point-to-Point Networks.” *Information & Computation* 207(8): 889–899, 2009.
19. I. Haitner, O. Horvitz, J. Katz, C.-Y. Koo, R. Morselli, and R. Shaltiel. “Reducing Complexity Assumptions for Statistically-Hiding Commitment.” *J. Cryptology* 22(3): 283–310, 2009.
20. A. Bender, J. Katz, and R. Morselli. “Ring Signatures: Stronger Definitions, and Constructions Without Random Oracles.” *J. Cryptology* 22(1): 114–138, 2009.
21. J. Katz and C.-Y. Koo. “On Expected Constant-Round Protocols for Byzantine Agreement.” *J. Computer and System Sciences* 75(2): 91–112, 2009.
22. J. Katz and Y. Lindell. “Handling Expected Polynomial-Time Strategies in Simulation-Based Security Proofs.” *J. Cryptology* 21(3): 303–349, 2008.
23. E.-J. Goh, S. Jarecki, J. Katz, and N. Wang. “Efficient Signature Schemes with Tight Security Reductions to the Diffie-Hellman Problems.” *J. Cryptology* 20(4): 493–514, 2007.
24. R. Canetti, S. Halevi, and J. Katz. “A Forward-Secure Public-Key Encryption Scheme.” *J. Cryptology* 20(3): 265–294, 2007.
25. J. Katz and M. Yung. “Scalable Protocols for Authenticated Group Key Exchange.” *J. Cryptology* 20(1): 85–113, 2007.
26. D. Boneh, R. Canetti, S. Halevi, and J. Katz. “Chosen-Ciphertext Security from Identity-Based Encryption.” *SIAM J. Computing* 36(5): 1301–1328, 2007.
27. J. Katz and M. Yung. “Characterization of Security Notions for Probabilistic Private-Key Encryption.” *J. Cryptology* 19(1): 67–96, 2006.
28. W. Du, J. Deng, Y.S. Han, P.K. Varshney, J. Katz, and A. Khalili. “A Pairwise Key Pre-Distribution Scheme for Wireless Sensor Networks.” *ACM Trans. on Information and System Security* 8(2): 228–258, 2005.
29. R. Gennaro, Y. Gertner, J. Katz, and L. Trevisan. “Bounds on the Efficiency of Generic Cryptographic Constructions.” *SIAM J. Computing* 35(1): 217–246, 2005.
30. J. Katz and Y. Lindell. “Aggregate Message Authentication Codes.” Accepted to *IET Information Security* (pending minor revisions).
31. J. Katz and C.-Y. Koo. “On Constructing Universal One-Way Hash Functions from Arbitrary One-Way Functions.” Accepted to *J. Cryptology* (pending minor revisions).

## Articles in Refereed Conferences and Workshops

1. C. Hong, J. Katz, V. Kolesnikov, W. Ju, and X. Wang. “Covert Security with Public Verifiability: Faster, Leaner, and Simpler.” *Advances in Cryptology—Eurocrypt 2019*.
2. D. Apon, D. Dachman-Soled, H. Gong, and J. Katz. “Constant-Round Group Key-Exchange from the Ring-LWE Assumption.” *Intl. Conference on Post-Quantum Cryptography 2019*.
3. N. Gupta, J. Katz, and N. Chopra. “Statistical Privacy in Distributed Average Consensus on Finite Real-Valued Inputs.” *American Control Conference 2019*.
4. D. Gordon, J. Katz, and X. Wang. “Simple and Efficient Two-Server ORAM.” *Advances in Cryptology—Asiacrypt 2018*.
5. H. Chan, J. Katz, K. Nayak, A. Polychroniadou, and E. Shi. “More is Less: Perfectly Secure Oblivious Algorithms in the Multi-Server Setting.” *Advances in Cryptology—Asiacrypt 2018*.
6. J. Katz, V. Kolesnikov, and X. Wang. “Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures.” *Proc. 25th ACM Conf. on Computer and Communications Security*, 2018.
7. J. Katz, S. Ranellucci, M. Rosulek, and X. Wang. “Optimizing Authenticated Garbling for Faster Secure Two-Party Computation.” *Advances in Cryptology—Crypto 2018*.
8. B. Cogliati, Y. Dodis, J. Katz, J. Lee, J. Steinberger, A. Thiruvengadam, and Z. Zhang. “Provable Security of (Tweakable) Block Ciphers Based on Substitution-Permutation Networks.” *Advances in Cryptology—Crypto 2018*.
9. Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, C. Papamanthou. “vRAM: Faster Verifiable RAM with Program-Independent Preprocessing.” *IEEE Symp. on Security & Privacy (Oakland) 2018*.
10. J. Katz, M. Maffei, G. Malavolta, and D. Schröder. “Subset Predicate Encryption and Its Applications.” *Cryptology and Network Security (CANS)*, 2017.
11. X. Wang, S. Ranellucci, and J. Katz. “Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation.” *Proc. 24th ACM Conf. on Computer and Communications Security*, 2017. **Recipient of best paper award.**
12. X. Wang, S. Ranellucci, and J. Katz. “Global-Scale Secure Multi-Party Computation.” *Proc. 24th ACM Conf. on Computer and Communications Security*, 2017.
13. D. Maimon, M. Becker, S. Patil, and J. Katz. “Self-Protective Behaviors over Public WiFi Networks.” *Learning from Authoritative Security Experiment Results (LASER)*, 2017.

14. C. Freitag, J. Katz, and N. Klein. "Symmetric-Key Broadcast Encryption: The Multi-Sender Case." *Intl. Symp. on Cyber Security, Cryptography, and Machine Learning 2017*.
15. D. Apon, C. Cho, K. Eldefrawy, and J. Katz. "Efficient, Reusable Fuzzy Extractors from LWE." *Intl. Symp. on Cyber Security, Cryptography, and Machine Learning 2017*.
16. Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. "vSQL: Verifying General SQL Queries over Dynamic Outsourced Databases." *IEEE Symp. on Security & Privacy (Oakland) 2017*.
17. X. Wang, A. Malozemoff, and J. Katz. "Faster Secure Two-Party Computation in the Single-Execution Setting." *Advances in Cryptology—Eurocrypt 2017*.
18. Y. Dodis, S. Guo, and J. Katz. "Fixing Cracks in the Concrete: Random Oracles with Auxiliary Input, Revisited." *Advances in Cryptology—Eurocrypt 2017*.
19. N. Gupta, J. Katz, and N. Chopra. "Privacy in Distributed Average Consensus." *20th International Federation of Automatic Control (IFAC) World Congress, 2017*.
20. K. Liao and J. Katz. "Incentivizing Blockchain Forks via Whale Transactions." *4th Workshop on Bitcoin and Blockchain Research, 2017*.
21. Y. Zhang, J. Katz, and C. Papamanthou. "An Expressive (Zero-Knowledge) Set Accumulator." *IEEE European Symposium on Security & Privacy 2017*.
22. V.T. Hoang, J. Katz, A. O'Neill, and M. Zaheri. "Selective-Opening Security in the Presence of Randomness Failures." *Advances in Cryptology—Asiacrypt 2016*.
23. J. Katz. "Analysis of a Proposed Hash-Based Signature Standard." *3rd International Conference on Research in Security Standardisation (SSR), 2016*.
24. K. Lewi, A. Malozemoff, D. Apon, B. Carmer, A. Foltzer, D. Wagner, D. Archer, D. Boneh, J. Katz, and M. Raykova. "5Gen: A Framework for Prototyping Applications Using Multilinear Maps and Matrix Branching Programs." *Proc. 23rd ACM Conf. on Computer and Communications Security, 2016*.
25. X. Wang, S.D. Gordon, A. McIntosh, and J. Katz. "Secure Computation of MIPS Machine Code." *ESORICS 2016*.
26. M.D. Green, J. Katz, A. Malozemoff, and H.-S. Zhou. "A Unified Approach to Idealized Model Separations via Indistinguishability Obfuscation." *10th Conf. on Security and Cryptography for Networks (SCN) 2016*.
27. Y. Zhang, J. Katz, and C. Papamanthou. "All Your Queries are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption." *USENIX Security Symposium 2016*.
28. R. Zhu, Y. Huang, J. Katz, and A. Shelat. "The Cut-and-Choose Game and Its Application to Cryptographic Protocols." *USENIX Security Symposium 2016*.

29. S. Zahur, X. Wang, M. Raykova, A. Gascon, J. Doerner, D. Evans, and J. Katz. “Revisiting Square Root ORAM: Efficient Random Access in Secure Computation.” *IEEE Symp. on Security & Privacy (Oakland)* 2016.
30. J. Katz, D. Dachman-Soled, and A. Thiruvengadam. “10-Round Feistel is Indifferentiable from an Ideal Cipher.” *Advances in Cryptology—Eurocrypt* 2016.
31. V.T. Hoang, J. Katz, and A. Malozemoff. “Automated Analysis and Synthesis of Authenticated Encryption Schemes.” *Proc. 22nd ACM Conf. on Computer and Communications Security*, 2015. **Recipient of best paper award.**
32. Y. Zhang, J. Katz, and C. Papamanthou. “IntegriDB: Verifiable SQL for Outsourced Databases.” *Proc. 22nd ACM Conf. on Computer and Communications Security*, 2015.
33. A. Miller, A. Kosba, J. Katz, and E. Shi. “Nonoutsourcable Scratch-Off Puzzles to Discourage Bitcoin Mining Coalitions.” *Proc. 22nd ACM Conf. on Computer and Communications Security*, 2015.
34. J. Garay, J. Katz, B. Tackman, and V. Zikas. “How Fair is Your Protocol? A Utility-Based Approach to Protocol Optimality.” *ACM Symposium on Principles of Distributed Computing (PODC)* 2015.
35. D. Dachman-Soled, J. Katz, and V. Rao. “Adaptively Secure, Universally Composable, Multi-Party Computation in Constant Rounds.” *12th Theory of Cryptography Conference (TCC)* 2015.
36. S.D. Gordon, J. Katz, F.-H. Liu, E. Shi, and H.-S. Zhou. “Multi-Client Verifiable Computation with Stronger Security Guarantees.” *12th Theory of Cryptography Conference (TCC)* 2015.
37. J. Katz, S. Lucks, and A. Thiruvengadam. “Hash Functions from Defective Ideal Ciphers.” *RSA Conference—Cryptographers’ Track* 2015.
38. Y. Zhang, C. Papamanthou, and J. Katz. “Alitheia: Towards Practical Verifiable Graph Processing.” *Proc. 21st ACM Conf. on Computer and Communications Security*, 2014.
39. Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. Malozemoff. “Amortizing Garbled Circuits.” *Advances in Cryptology—Crypto* 2014.
40. D. Dachman-Soled, N. Fleischhacker, J. Katz, Anna Lysyanskaya, and Dominique Schroöder. “Feasibility and Infeasibility of Secure Computation with Malicious PUFs.” *Advances in Cryptology—Crypto* 2014.
41. S.G. Choi, J. Katz, A. Malozemoff, and V. Zikas. “Efficient Three-Party Computation from Cut-and-Choose.” *Advances in Cryptology—Crypto* 2014.
42. A. Malozemoff, J. Katz, and M. Green. “Automated Analysis and Synthesis of Block-Cipher Modes of Operation.” *IEEE Computer Security Foundations Symposium* 2014.

43. J. Katz, A. Kiayias, H.-S. Zhou, and V. Zikas. “Distributing the Setup in Universally Composable Multiparty Computation.” *ACM Symposium on Principles of Distributed Computing (PODC)* 2014.
44. C. Liu, Y. Huang, E. Shi, J. Katz, and M. Hicks. “Automating Efficient RAM-Model Secure Computation.” *IEEE Symp. on Security & Privacy (Oakland)* 2014.
45. A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. “PermaCoin: Repurposing Bitcoin Work for Long-Term Data Preservation.” *IEEE Symp. on Security & Privacy (Oakland)* 2014.
46. S. Goldwasser, S.D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou. “Multi-Input Functional Encryption.” *Advances in Cryptology—Eurocrypt 2014*.
47. D. Apon, J. Katz, E. Shi, and A. Thiruvengadam. “Verifiable Oblivious Storage.” *Public-Key Cryptography (PKC)* 2014.
48. S.G. Choi, J. Katz, D. Schröder, A. Yerukhimovich, and H.-S. Zhou. “(Efficient) Universally Composable Oblivious Transfer with a Minimal Number of Stateless Tokens.” *11th Theory of Cryptography Conference (TCC) 2014*, pp. 638–662, LNCS vol. 8349, Springer, 2014. **One of three papers invited to *J. Cryptology*.**
49. A. Miller, M. Hicks, J. Katz, and E. Shi. “Authenticated Data Structures, Generically.” *ACM Symp. on Principles of Programming Languages (POPL)* 2014.
50. K.-M. Chung, J. Katz, and H.-S. Zhou. “Functional Encryption from (Small) Hardware Tokens.” *Advances in Cryptology—Asiacrypt 2013*.
51. J. Garay, J. Katz, U. Maurer, B. Tackmann, and V. Zikas. “Rational Protocol Design: Cryptography Against Incentive-Driven Attackers.” *Proc. 54th Annual Symposium on Foundations of Computer Science (FOCS)*, 2013.
52. R. Bassily, A. Groce, J. Katz, and A. Smith. “Coupled-Worlds Privacy: Exploiting Adversarial Uncertainty in Statistical Data Privacy.” *Proc. 54th Annual Symposium on Foundations of Computer Science (FOCS)*, 2013.
53. Y. Huang, J. Katz, and D. Evans. “Efficient Secure Two-Party Computation Using Symmetric Cut-and-Choose.” *Advances in Cryptology—Crypto 2013*.
54. M. Lee, A. Dunn, J. Katz, B. Waters, and E. Witchel. “Anon-Pass: Practical Anonymous Subscriptions.” *IEEE Symp. on Security & Privacy (Oakland)* 2013. **Invited to a special issue of *IEEE Security & Privacy* magazine.**
55. S.G. Choi, J. Katz, R. Kumaresan, and C. Cid. “Multi-Client Non-Interactive Verifiable Computation.” *10th Theory of Cryptography Conference (TCC)* 2013.
56. S. Fehr, J. Katz, F. Song, H.-S. Zhou, and V. Zikas. “Feasibility and Completeness of Cryptographic Tasks in the Quantum World.” *10th Theory of Cryptography Conference (TCC)* 2013.



57. J. Katz, U. Maurer, B. Tackmann, and V. Zikas. “Universally Composable Synchronous Computation.” *10th Theory of Cryptography Conference (TCC) 2013*.
58. S.G. Choi, J. Katz, H. Wee, and H.-S. Zhou. “Efficient, Adaptively Secure, and Composable Oblivious Transfer with a Single, Global CRS.” *Public-Key Cryptography (PKC) 2013*.
59. J. Katz, A. Thiruvengadam, and H.-S. Zhou. “Feasibility and Infeasibility of Adaptively Secure, Fully Homomorphic Encryption.” *Public-Key Cryptography (PKC) 2013*.
60. D. Gordon, J. Katz, V. Kolesnikov, F. Krell, T. Malkin, M. Raykova, and Y. Vahlis. “Secure Two-Party Computation in Sublinear Amortized Time.” *Proc. 19th ACM Conf. on Computer and Communications Security*, 2012.
61. J. Alwen, J. Katz, U. Maurer, and V. Zikas. “Collusion-Preserving Computation.” *Advances in Cryptology—Crypto 2012*.
62. A. Groce, J. Katz, A. Thiruvengadam, and V. Zikas. “Byzantine Agreement with a Rational Adversary.” *Intl. Colloquium on Automata, Languages, and Programming (ICALP) 2012*, pp. 561–572, LNCS vol. 7392, Springer, 2012.
63. P. Mardziel, M. Hicks, J. Katz, and M. Srivatsa. “Knowledge-Oriented Secure Multiparty Computation.” *ACM Workshop on Programming and Analysis for Security (PLAS) 2012*.
64. Y. Huang, J. Katz, and D. Evans. “Quid Pro Quo-tocols: Strengthening Semi-Honest Protocols with Dual Execution.” *IEEE Symp. on Security & Privacy (Oakland) 2012*.
65. J.H. Seo, J.H. Cheon, and J. Katz. “Constant-Round Multi-Party Private Set Union Using Reversed Laurent Series.” *Public-Key Cryptography (PKC) 2012*, pp. 398–412, LNCS vol. 7293, Springer, 2012.
66. A. Groce and J. Katz. “Fair Computation with Rational Players.” *Advances in Cryptology—Eurocrypt 2012*, pp. 81–98, LNCS vol. 7237, Springer, 2012.
67. S.G. Choi, J. Katz, R. Kumaresan, and H.-S. Zhou. “On the Security of the ‘Free-XOR’ Technique.” *9th Theory of Cryptography Conference (TCC) 2012*, pp. 39–53, LNCS vol. 7194, Springer, 2012.
68. S.G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D. Rubenstein. “Secure Multi-Party Computation of Boolean Circuits with Applications to Privacy in On-Line Marketplaces.” *RSA Conference—Cryptographers’ Track 2012*, pp. 416–432, LNCS vol. 7178, Springer, 2012.
69. Y. Huang, D. Evans, and J. Katz. “Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?” *Network and Distributed System Security Conference (NDSS) 2012*.

70. Y. Huang, C.-H. Shen, D. Evans, J. Katz, and A. Shelat. “Efficient Secure Computation with Garbled Circuits” (invited paper). *Intl. Conference on Information Systems Security (ICISS)*, pp. 28–48, LNCS vol. 7093, Springer, 2011.
71. J. Katz and L. Malka. “Constant-Round Private-Function Evaluation with Linear Complexity.” *Advances in Cryptology—Asiacrypt 2011*, pp. 556–571, LNCS vol. 7073, Springer, 2011.
72. Y. Huang, D. Evans, J. Katz, and L. Malka. “Faster Secure Two-Party Computation Using Garbled Circuits.” *20th USENIX Security Symposium*, 2011.
73. J. Garay, J. Katz, R. Kumaresan, and H.-S. Zhou. “Adaptively Secure Broadcast, Revisited.” *ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 179–186, ACM, 2011.
74. J. Katz and V. Vaikuntanathan. “Round-Optimal Password-Based Authenticated Key Exchange.” *8th Theory of Cryptography Conference (TCC)*, pp. 293–310, LNCS vol. 6597, Springer, 2011. **One of three papers invited to *J. Cryptology*.**
75. A. Groce, J. Katz, and A. Yerukhimovich. “Limits of Computational Differential Privacy in the Client/Server Setting.” *8th Theory of Cryptography Conference (TCC)*, pp. 417–431, LNCS vol. 6597, Springer, 2011.
76. Z. Brakerski, J. Katz, G. Segev, and A. Yerukhimovich. “Limits on the Power of Zero-Knowledge Proofs in Cryptographic Constructions.” *8th Theory of Cryptography Conference (TCC)*, pp. 559–578, LNCS vol. 6597, Springer, 2011.
77. J. Katz, D. Schröder, and A. Yerukhimovich. “Impossibility of Blind Signatures from One-Way Permutations.” *8th Theory of Cryptography Conference (TCC)*, pp. 615–629, LNCS vol. 6597, Springer, 2011.
78. Y. Huang, L. Malka, D. Evans, and J. Katz. “Efficient Privacy-Preserving Biometric Identification.” *Network & Distributed System Security Conference (NDSS) 2011*.
79. S.D. Gordon, J. Katz, and V. Vaikuntanathan. “A Group Signature Scheme from Lattice Assumptions.” *Advances in Cryptology—Asiacrypt 2010*, pp. 395–412, LNCS vol. 6477, Springer, 2010.
80. Z. Brakerski, Y. Tauman Kalai, J. Katz, and V. Vaikuntanathan. “Public-Key Cryptography Resilient to Continual Memory Leakage.” *Proc. 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 501–510, IEEE, 2010.
81. J. Katz and L. Malka. “Secure Text Processing with Applications to Private DNA Matching.” *Proc. 17th ACM Conf. on Computer and Communications Security*, pp. 485–492, ACM, 2010.
82. A. Groce and J. Katz. “A New Framework for Efficient Password-Based Authenticated Key Exchange.” *Proc. 17th ACM Conf. on Computer and Communications Security*, pp. 516–525, ACM, 2010.

83. S.D. Gordon, J. Katz, R. Kumaresan, and A. Yerukhimovich. “Authenticated Broadcast with a Partially Compromised Public-Key Infrastructure.” *12th Intl. Symp. on Stabilization, Safety, and Security of Distributed Systems*, pp. 144–158, LNCS vol. 6366, Springer, 2010. **Invited to a special issue of *Information & Computation*.**
84. D. Gordon and J. Katz. “Partial Fairness in Secure Computation.” *Advances in Cryptology—Eurocrypt 2010*, pp. 157–176, LNCS vol. 6110, Springer, 2010.
85. R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin. “Secure Network Coding over the Integers.” *Public-Key Cryptography (PKC)*, pp. 142–160, LNCS vol. 6056, Springer, 2010.
86. G. Fuchsbauer, J. Katz, and D. Naccache. “Efficient Rational Secret Sharing in Standard Communication Networks.” *7th Theory of Cryptography Conference (TCC)*, pp. 419–436, LNCS vol. 5978, Springer, 2010.
87. J. Katz and V. Vaikuntanathan. “Signature Schemes with Bounded Leakage Resilience.” *Advances in Cryptology—Asiacrypt 2009*, pp. 703–720, LNCS vol. 5912, Springer, 2009.
88. J. Katz and A. Yerukhimovich. “On Black-Box Constructions of Predicate Encryption Schemes from Trapdoor Permutations.” *Advances in Cryptology—Asiacrypt 2009*, pp. 197–213, LNCS vol. 5912, Springer, 2009.
89. J. Katz and V. Vaikuntanathan. “Smooth Projective Hashing and Password-Based Authenticated Key Exchange from Lattices.” *Advances in Cryptology—Asiacrypt 2009*, pp. 636–652, LNCS vol. 5912, Springer, 2009.
90. G. Ateniese, S. Kamara, and J. Katz. “Proofs of Storage from Homomorphic Identification Protocols.” *Advances in Cryptology—Asiacrypt 2009*, pp. 319–333, LNCS vol. 5912, Springer, 2009.
91. M. Albrecht, C. Gentry, S. Halevi, and J. Katz. “Attacking Cryptographic Schemes Based on ‘Perturbation Polynomials’.” *Proc. 16th ACM Conf. on Computer and Communications Security*, pp. 1–10, ACM, 2009.
92. J. Alwen, J. Katz, Y. Lindell, G. Persiano, A. Shelat, and I. Visconti. “Collusion-Free Multiparty Computation in the Mediated Model.” *Advances in Cryptology—Crypto 2009*, pp. 524–540, LNCS vol. 5677, Springer, 2009.
93. D. Boneh, J. Katz, D. Freeman, and B. Waters. “Signing a Linear Subspace: Signatures for Network Coding.” *Public-Key Cryptography (PKC)*, pp. 68–87, LNCS vol. 5443, Springer, 2009.
94. Y. Dodis, J. Katz, A. Smith, and S. Walfish. “Composability and On-Line Deniability of Authentication.” *6th Theory of Cryptography Conference (TCC)*, pp. 146–162, LNCS vol. 5444, Springer, 2009.

95. S.D. Gordon and J. Katz. “Complete Fairness in Multi-Party Computation Without an Honest Majority.” *6th Theory of Cryptography Conference (TCC)*, pp. 19–35, LNCS vol. 5444, Springer, 2009.
96. J. Katz, C.-Y. Koo, and R. Kumaresan. “Improving the Round Complexity of VSS in Point-to-Point Networks.” *Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, pp. 499–510, LNCS vol. 5126, Springer, 2008.
97. S.D. Gordon, C. Hazay, J. Katz, and Y. Lindell. “Complete Fairness in Secure Two-Party Computation.” *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC) 2008*, pp. 413–422, ACM, 2008.
98. J. Katz, A. Sahai, and B. Waters. “Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products.” *Advances in Cryptology—Eurocrypt 2008*, pp. 146–162, LNCS vol. 4965, Springer, 2008. **One of four papers invited to J. Cryptology.**
99. S. Kamara and J. Katz. “How to Encrypt with a Malicious Random Number Generator.” *Fast Software Encryption (FSE)*, pp. 303–315, LNCS vol. 5086, Springer, 2008.
100. J. Katz and Y. Lindell. “Aggregate Message Authentication Codes.” *RSA Conference—Cryptographers’ Track*, pp. 155–169, LNCS vol. 4964, Springer, 2008.
101. J. Katz. “Bridging Cryptography and Game Theory: Recent Results and Future Directions” (invited paper). *5th Theory of Cryptography Conference (TCC)*, pp. 251–272, LNCS vol. 4948, Springer, 2008.
102. J. Katz. “Which Languages Have 4-Round Zero-Knowledge Proofs?” *5th Theory of Cryptography Conference (TCC)*, pp. 73–88, LNCS vol. 4948, Springer, 2008. **One of three papers invited to J. Cryptology.**
103. V. Goyal and J. Katz. “Universally-Composable Computation with an Unreliable Common Reference String.” *5th Theory of Cryptography Conference (TCC)*, pp. 142–154, LNCS vol. 4948, Springer, 2008.
104. J. Katz. “Efficient Cryptographic Protocols Based on the Hardness of Learning Parity with Noise” (invited paper). *11th IMA Intl. Conference on Cryptography and Coding Theory*, pp. 1–15, Lecture Notes in Computer Science vol. 4887, Springer, 2007.
105. J. Garay, J. Katz, C.-Y. Koo, and R. Ostrovsky. “Round Complexity of Authenticated Broadcast with a Dishonest Majority.” *Proc. 48th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 658–668, IEEE, 2007.
106. O. Horvitz and J. Katz. “Universally Composable Two-Party Computation in Two Rounds.” *Advances in Cryptology—Crypto 2007*, pp. 111–129, Lecture Notes in Computer Science vol. 4622, Springer, 2007.

107. R. Morselli, B. Bhattacharjee, J. Katz, and M. Marsh. "Exploiting Approximate Transitivity of Trust" (invited paper). *4th Intl. Conf. on Broadband Communications, Networks, and Systems (BroadNets)*, pp. 515–524, IEEE, 2007.
108. J. Katz. "On Achieving the 'Best of Both Worlds' in Secure Multiparty Computation." *Proc. 39th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 11–20, ACM, 2007.
109. J. Katz. "Universally-Composable Multi-Party Computation using Tamper-Proof Hardware." *Advances in Cryptology—Eurocrypt 2007*, pp. 115–128, Lecture Notes in Computer Science vol. 4515, Springer, 2007.
110. J. Katz and C.-Y. Koo. "Round-Efficient Secure Computation in Point-to-Point Networks." *Advances in Cryptology—Eurocrypt 2007*, pp. 311–328, Lecture Notes in Computer Science vol. 4515, Springer, 2007.
111. C. Hazay, J. Katz, C.-Y. Koo, and Y. Lindell. "Concurrently-Secure Blind Signatures without Random Oracles or Setup Assumptions." *4th Theory of Cryptography Conference (TCC)*, pp. 323–341, Lecture Notes in Computer Science vol. 4391, Springer, 2007.
112. S.D. Gordon and J. Katz. "Rational Secret Sharing, Revisited." *Security and Cryptography for Networks (SCN)*, pp. 229–241, Lecture Notes in Computer Science vol. 4116, Springer, 2006. An extended abstract of this work also appeared at *NetEcon 2006*.
113. J. Katz and C.-Y. Koo. "On Expected Constant-Round Protocols for Byzantine Agreement." *Advances in Cryptology—Crypto 2006*, pp. 445–462, Lecture Notes in Computer Science vol. 4117, Springer, 2006.
114. Y. Dodis, J. Katz, L. Reyzin, and A. Smith. "Authenticated Key Agreement from 'Close' Secrets." *Advances in Cryptology—Crypto 2006*, pp. 232–250, Lecture Notes in Computer Science vol. 4117, Springer, 2006.
115. C.-Y. Koo, V. Bhandari, J. Katz, and N. Vadiya. "Reliable Broadcast in Radio Networks: The Bounded Collision Case." *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 258–262, ACM, 2006.
116. J. Katz and J.S. Shin. "Parallel and Concurrent Security of the HB and HB<sup>+</sup> Protocols." *Advances in Cryptology—Eurocrypt 2006*, pp. 73–87, Lecture Notes in Computer Science vol. 4004, Springer, 2006.
117. A. Bender, J. Katz, and R. Morselli. "Ring Signatures: Stronger Definitions, and Constructions without Random Oracles." *3rd Theory of Cryptography Conference (TCC)*, pp. 60–79, Lecture Notes in Computer Science vol. 3876, Springer, 2006.
118. J. Katz and J.S. Shin. "Modeling Insider Attacks on Group Key-Exchange Protocols." *Proc. 12th ACM Conf. on Computer and Communications Security*, pp. 180–189, ACM, 2005.

119. O. Horvitz and J. Katz. “Lower Bounds on the Efficiency of ‘Black-Box’ Commitment Schemes.” *International Colloquium on Automata, Languages, and Programming (ICALP)*, pp. 128–139, Lecture Notes in Computer Science vol. 3580, Springer, 2005. **Invited to a special issue of *Theoretical Computer Science*.**
120. J. Katz, P. MacKenzie, G. Taban, and V. Gligor. “Two-Server Password-Only Authenticated Key Exchange.” *Applied Cryptography and Network Security (ACNS)*, pp. 1–16, Lecture Notes in Computer Science vol. 3531, Springer, 2005.
121. X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith. “Secure Remote Authentication Using Biometric Data.” *Advances in Cryptology—Eurocrypt 2005*, pp. 147–163, Lecture Notes in Computer Science vol. 3494, Springer, 2005.
122. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. “Universally Composable Password-Based Key Exchange.” *Advances in Cryptology—Eurocrypt 2005*, pp. 404–421, Lecture Notes in Computer Science vol. 3494, Springer, 2005.
123. I. Haitner, O. Horvitz, J. Katz, C.-Y. Koo, R. Morselli, and R. Shaltiel. “Reducing Complexity Assumptions for Statistically-Hiding Commitment.” *Advances in Cryptology—Eurocrypt 2005*, pp. 58–77, Lecture Notes in Computer Science vol. 3494, Springer, 2005.
124. R. Canetti, S. Halevi, and J. Katz. “Adaptively-Secure, Non-Interactive Public-Key Encryption.” *2nd Theory of Cryptography Conference (TCC)*, pp. 150–168, Lecture Notes in Computer Science vol. 3378, Springer, 2005.
125. J. Katz and Y. Lindell. “Handling Expected Polynomial-Time Strategies in Simulation Based Security Proofs.” *2nd Theory of Cryptography Conference (TCC)*, pp. 128–149, Lecture Notes in Computer Science vol. 3378, Springer, 2005.
126. Y. Dodis and J. Katz. “Chosen-Ciphertext Security of Multiple Encryption.” *2nd Theory of Cryptography Conference (TCC)*, pp. 188–209, Lecture Notes in Computer Science vol. 3378, Springer, 2005.
127. D. Boneh and J. Katz. “Improved Efficiency for CCA-Secure Cryptosystems Built Using Identity-Based Encryption.” *RSA Conference—Cryptographers’ Track*, pp. 87–103, Lecture Notes in Computer Science vol. 3376, Springer, 2005.
128. J. Katz, R. Ostrovsky, and M.O. Rabin. “Identity-Based Zero Knowledge.” *Security in Communication Networks (SCN)*, pp. 180–192, Lecture Notes in Computer Science vol. 3352, Springer, 2004.
129. R. Morselli, J. Katz, and B. Bhattacharjee. “A Game-Theoretic Framework for Analyzing Trust-Inference Protocols.” *Second Workshop on the Economics of Peer-to-Peer Systems*, Boston, MA, 2004.
130. J. Katz and R. Ostrovsky. “Round-Optimal Secure Two-Party Computation.” *Advances in Cryptology—Crypto 2004*, pp. 335–354, Lecture Notes in Computer Science vol. 3152, Springer, 2004.

131. I.R. Jeong, J. Katz, D.H. Lee. “One-Round Protocols for Two-Party Authenticated Key Exchange.” *Applied Cryptography and Network Security (ACNS)*, pp. 220–232, Lecture Notes in Computer Science vol. 3089, Springer, 2004.
132. R. Canetti, S. Halevi, and J. Katz. “Chosen-Ciphertext Security from Identity-Based Encryption.” *Advances in Cryptology—Eurocrypt 2004*, pp. 207–222, Lecture Notes in Computer Science vol. 3027, Springer, 2004.
133. R. Morselli, B. Bhattacharjee, J. Katz, and P. Keleher. “Trust-Preserving Set Operations.” *Proc. IEEE INFOCOM*, pp. 2231–2241, IEEE, 2004.
134. Y. Dodis, M. Franklin, J. Katz, A. Miyaji, and M. Yung. “A Generic Construction for Intrusion-Resilient Public-Key Encryption.” *RSA Conference—Cryptographers’ Track*, pp. 81–98, Lecture Notes in Computer Science vol. 2964, Springer, 2004.
135. J. Katz. “Binary Tree Encryption: Constructions and Applications” (invited paper). *6th Intl. Conference on Information Security and Cryptology (ICISC)*, pp. 1–11, Lecture Notes in Computer Science vol. 2971, Springer, 2003.
136. J. Katz and N. Wang. “Efficiency Improvements for Signature Schemes with Tight Security Reductions.” *Proc. 10th ACM Conf. on Computer and Communications Security*, pp. 155–164, ACM, 2003.
137. J. Katz and M. Yung. “Scalable Protocols for Authenticated Group Key Exchange.” *Advances in Cryptology—Crypto 2003*, pp. 110–125, Lecture Notes in Computer Science vol. 2729, Springer, 2003.
138. R. Gennaro, Y. Gertner, and J. Katz. “Lower Bounds on the Efficiency of Encryption and Digital Signature Schemes.” *Proc. 35th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 417–425, ACM, 2003.
139. J. Katz, R. Ostrovsky, and A. Smith. “Round Efficiency of Multi-Party Computation with Dishonest Majority.” *Advances in Cryptology—Eurocrypt 2003*, pp. 578–595, Lecture Notes in Computer Science vol. 2656, Springer, 2003.
140. R. Canetti, S. Halevi, and J. Katz. “A Forward-Secure Public-Key Encryption Scheme.” *Advances in Cryptology—Eurocrypt 2003*, pp. 255–272, Lecture Notes in Computer Science vol. 2656, Springer, 2003.
141. J. Katz. “Efficient and Non-Malleable Proofs of Plaintext Knowledge and Applications.” *Advances in Cryptology—Eurocrypt 2003*, pp. 211–228, Lecture Notes in Computer Science vol. 2656, Springer, 2003.
142. A. Khalili, J. Katz, and W. Arbaugh. “Toward Secure Key Distribution in Truly Ad-Hoc Networks.” *2003 Symposium on Applications and the Internet Workshops*, pp. 342–346, IEEE, 2003.
143. Y. Dodis, M. Franklin, J. Katz, A. Miyaji, and M. Yung. “Intrusion-Resilient Public-Key Encryption.” *RSA Conference—Cryptographers’ Track*, pp. 19–32, Lecture Notes in Computer Science vol. 2612, Springer, 2003.

144. Y. Dodis, J. Katz, S. Xu, and M. Yung. “Strong Key-Insulated Signature Schemes.” *Public-Key Cryptography (PKC)*, pp. 130–144, Lecture Notes in Computer Science vol. 2567, Springer, 2003.
145. J. Katz, R. Ostrovsky, and M. Yung. “Forward Secrecy in Password-Only Key-Exchange Protocols.” *Security in Communication Networks (SCN)*, pp. 29–44, Lecture Notes in Computer Science vol. 2576, Springer, 2002.
146. J. Katz and M. Yung. “Threshold Cryptosystems Based on Factoring.” *Advances in Cryptology—Asiacrypt 2002*, pp. 192–205, Lecture Notes in Computer Science vol. 2501, Springer, 2002.
147. K. Jallad, J. Katz, and B. Schneier. “Implementation of Chosen-Ciphertext Attacks against PGP and GnuPG.” *Information Security Conference*, pp. 90–101, Lecture Notes in Computer Science vol. 2433, Springer, 2002.
148. Y. Dodis, J. Katz, S. Xu, and M. Yung. “Key-Insulated Public-Key Cryptosystems.” *Advances in Cryptology—Eurocrypt 2002*, pp. 65–82, Lecture Notes in Computer Science vol. 2332, Springer, 2002.
149. E. Buonanno, J. Katz, and M. Yung. “Incremental and Unforgeable Encryption.” *Fast Software Encryption (FSE)*, pp. 109–124, Lecture Notes in Computer Science vol. 2355, Springer, 2002.
150. J. Katz, R. Ostrovsky, and M. Yung. “Efficient Password-Authenticated Key-Exchange Using Human-Memorizable Passwords.” *Advances in Cryptology—Eurocrypt 2001*, pp. 474–494, Lecture Notes in Computer Science vol. 2045, Springer, 2001.
151. J. Katz, R. Ostrovsky, and S. Myers. “Cryptographic Counters and Applications to Electronic Voting.” *Advances in Cryptology—Eurocrypt 2001*, pp. 78–92, Lecture Notes in Computer Science vol. 2045, Springer, 2001.
152. G. Di Crescenzo, J. Katz, R. Ostrovsky, and A. Smith. “Efficient and Non-Interactive, Non-Malleable Commitment.” *Advances in Cryptology—Eurocrypt 2001*, pp. 40–59, Lecture Notes in Computer Science vol. 2045, Springer, 2001.
153. J. Katz and B. Schneier. “A Chosen-Ciphertext Attack Against Several E-mail Encryption Protocols.” *Proc. 9th USENIX Security Symposium*, pp. 241–246, USENIX, 2000.
154. J. Katz and M. Yung. “Unforgeable Encryption and Chosen-Ciphertext-Secure Modes of Operation.” *Fast Software Encryption (FSE)*, pp. 284–299, Lecture Notes in Computer Science vol. 1978, Springer, 2001.
155. J. Katz and M. Yung. “Complete Characterization of Security Notions for Probabilistic, Private-Key Encryption.” *Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pp. 245–254, ACM, 2000.



156. J. Katz and L. Trevisan. “On the Efficiency of Local Decoding Procedures for Error-Correcting Codes.” *Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pp. 80–86, ACM, 2000.

## Other

1. S. Stevens et al. “Quantum Computing Risks to the Financial Services Industry.” Informative Report (ASC X9 IR 01-2019), ANSI ASC X9 Quantum Computing Risk Study Group, 2019.
2. Mark Flood, Jonathan Katz, Stephen Ong, and Adam Smith. “Cryptography and the Economics of Supervisory Information: Balancing Transparency and Confidentiality.” *Financial Stability Conference: Using the Tools, Finding the Data (2013)*.
3. Rajesh Chitnis, MohammadTaghi Hajiaghayi, Jonathan Katz, and Koyel Mukherjee. “A Game-Theoretic Model Motivated by the DARPA Network Challenge.” *SPAA 2013 brief announcement*. Also presented at the *Workshop on Risk Aversion in Algorithmic Game Theory and Mechanism Design, 2012*.
4. R. Morselli, B. Bhattacharjee, J. Katz, and M. Marsh, “KeyChains: A Decentralized Public-Key Infrastructure,” Technical Report CS-TR-4788, University of Maryland Computer Science Department, March, 2006.

## Patents/Patent Applications

- Karim El Defrawy, Chongwon Cho, Daniel C. Apon, and Jonathan Katz. Non-malleable Obfuscator for Sparse Functions. US Patent 10,198,584, Feb. 5, 2019.
- Chongwon Cho, Karim El Defrawy, Daniel C. Apon, and Jonathan Katz. Practical Reusable Fuzzy Extractor Based on the Learning-with-Error Assumption and Random Oracle. Application US 15/976,583. (Priority date July 17, 2017.)

## Expert Testimony (Last 5 Years)

Testifying expert for Case #2:18-cv-01844 for Patent 7,372,961: Blackberry Limited v. Facebook, Inc., WhatsApp Inc., and Instagram Inc., United District Court for the Central District of California. (Current)

Testifying expert for ZitoVault, LLC for IPR of Patent 6,484,257: Amazon.com Inc. et al. v. ZitoVault, LLC, IPR2016-00021. (2016)

Testifying expert for Secure Axxess, LLC for CBM of Patent 7,631,191: T. Rowe Price Investment Services v. Secure Axxess, LLC, CBM2015-00027. (2015)

Testifying expert for Smartflash, LLC for CBM of Patent 8,118,221: Apple Inc. v. Smartflash, LLC, CBM2014-00102, -00106, -00108, and -00112. (2015)

Testifying expert for Secure Axxess, LLC for CBM of Patent 7,631,191: PNC Bank et al. v. Secure Axxess, LLC, CBM2014-00100. (2014–2015)

Testifying expert for Secure Axxess, LLC for IPR of Patent 7,631,191: EMC Corporation and RSA Security, LLC v. Secure Axxess, LLC, IPR2014-00475. (2014–2015)