Introduction

601.642/442: Modern Cryptography

Fall 2020

What is Cryptography?

- **Controlling access to information**

What is Cryptography?

- **Controlling access to information**
  - "Who" learns "what?"

What is Cryptography?

- **Controlling access to information**
  - "Who" learns "what?"
  - "Who" can influence "what?"

What is Cryptography?

- **Controlling access to information**
  - "Who" learns "what?"
  - "Who" can influence "what?"
- **Relation to other areas**

What is Cryptography?

- **Controlling access to information**
  - "Who" learns "what?"
  - "Who" can influence "what?"
- **Relation to other areas**
  - Mathematical foundation of Information Security

What is Cryptography?

- **Controlling access to information**
  - "Who" learns "what?"
  - "Who" can influence "what?"
- **Relation to other areas**
  - Mathematical foundation of Information Security
  - Large intersection with: complexity theory, information theory, number theory, linear algebra, combinatorics...

A Brief History of Time

- Early forms of cryptography: Various kinds of *Ciphers* for secret communication. Today they are called *encryption schemes*.

A Brief History of Time

- Early forms of cryptography: Various kinds of *Ciphers* for secret communication. Today they are called *encryption schemes*.
- Many, many uses throughout history:

A Brief History of Time

- Early forms of cryptography: Various kinds of *Ciphers* for secret communication. Today they are called *encryption schemes*.
- Many, many uses throughout history:
  - Caesar Cipher (first used by Roman army in 1st century AD; broken in 5-9th century AD)

A Brief History of Time

- Early forms of cryptography: Various kinds of *Ciphers* for secret communication. Today they are called *encryption schemes*.
- Many, many uses throughout history:
  - Caesar Cipher (first used by Roman army in 1st century AD; broken in 5-9th century AD)
  - Vigenere Cipher (first used in Italy in 1460s; broken in 1800s)

A Brief History of Time

- Early forms of cryptography: Various kinds of *Ciphers* for secret communication. Today they are called *encryption schemes*.
- Many, many uses throughout history:
  - Caesar Cipher (first used by Roman army in 1st century AD; broken in 5-9th century AD)
  - Vigenere Cipher (first used in Italy in 1460s; broken in 1800s)
  - Enigma Code (invented by Germans and used by militaries around the world during World War II. Broken by Alan Turing and his team)

A Brief History of Time

- Early forms of cryptography: Various kinds of *Ciphers* for secret communication. Today they are called *encryption schemes*.
- Many, many uses throughout history:
  - Caesar Cipher (first used by Roman army in 1st century AD; broken in 5-9th century AD)
  - Vigenere Cipher (first used in Italy in 1460s; broken in 1800s)
  - Enigma Code (invented by Germans and used by militaries around the world during World War II. Broken by Alan Turing and his team)
  - ...

A Brief History of Time

- Early forms of cryptography: Various kinds of *Ciphers* for secret communication. Today they are called *encryption schemes*.
- Many, many uses throughout history:
  - Caesar Cipher (first used by Roman army in 1st century AD; broken in 5-9th century AD)
  - Vigenere Cipher (first used in Italy in 1460s; broken in 1800s)
  - Enigma Code (invented by Germans and used by militaries around the world during World War II. Broken by Alan Turing and his team)
  - ...
- Local history: Edgar Allan Poe was very interested in cryptography! E.g., read his book *The Gold Bug* from 1843 whose plot revolves around a cipher about a buried treasure.

A Brief History of Time (contd.)

- Fighting Apartheid: African National Congress was banned in South Africa 1960, after which it started to operate internationally. Because of the ban, they needed to communicate securely. They built their own secure communication system *Vula* to communicate between London, Zambia, Netherlands and South Africa. While they had computers, internet wasn't a thing then. So their communication system also used tape-recorders and public phone booths.

A Brief History of Time (contd.)

- Modern Cryptography started in the 1970s.

A Brief History of Time (contd.)

- Modern Cryptography started in the 1970s.
- The first annual international cryptology conference – CRYPTO – was held in 1981 with 102 attendees. **8 of them have since won the Turing Award**. (10 cryptographers in total; *so far*.)

A Brief History of Time (contd.)

- Modern Cryptography started in the 1970s.
- The first annual international cryptology conference – CRYPTO – was held in 1981 with 102 attendees. **8 of them have since won the Turing Award**. (10 cryptographers in total; *so far*.)
- Today, cryptography used in every day life. SSL, credit cards, ...

A Brief History of Time (contd.)

- Modern Cryptography started in the 1970s.
- The first annual international cryptology conference – CRYPTO – was held in 1981 with 102 attendees. **8 of them have since won the Turing Award**. (10 cryptographers in total; *so far*.)
- Today, cryptography used in every day life. SSL, credit cards, ...
- Note: Cryptocurrencies are NOT "crypto;" although they do use a lot of crypto (i.e., cryptography).

A Brief History of Time (contd.)

- Modern Cryptography started in the 1970s.
- The first annual international cryptology conference – CRYPTO – was held in 1981 with 102 attendees. **8 of them have since won the Turing Award**. (10 cryptographers in total; *so far*.)
- Today, cryptography used in every day life. SSL, credit cards, ...
- Note: Cryptocurrencies are NOT "crypto;" although they do use a lot of crypto (i.e., cryptography).
- Many emerging areas in Computer Science have roots in cryptography. A few prominent examples include Differential Privacy, Secure Machine Learning, and more.

A Brief History of Time (contd.)

- Modern Cryptography started in the 1970s.
- The first annual international cryptology conference – CRYPTO – was held in 1981 with 102 attendees. **8 of them have since won the Turing Award**. (10 cryptographers in total; *so far*.)
- Today, cryptography used in every day life. SSL, credit cards, ...
- Note: Cryptocurrencies are NOT "crypto;" although they do use a lot of crypto (i.e., cryptography).
- Many emerging areas in Computer Science have roots in cryptography. A few prominent examples include Differential Privacy, Secure Machine Learning, and more.
- Much of Cryptography relies on the intractability of certain "hard problems". The emergence of quantum computers poses a threat to some of these problems; as such (post)-quantum cryptography becomes increasingly important.

Course Objectives

Course Objectives

- Cryptography is full of *seeming* paradoxes. To develop confidence in any designed scheme, it is important to *prove* its security. The primary goal of this course is to learn the modern, provable security based approach to cryptography

Course Objectives

- Cryptography is full of *seeming* paradoxes. To develop confidence in any designed scheme, it is important to *prove* its security. The primary goal of this course is to learn the modern, provable security based approach to cryptography
- Introduce some of the core topics and some "hot areas"

Course Objectives

- Cryptography is full of *seeming* paradoxes. To develop confidence in any designed scheme, it is important to *prove* its security. The primary goal of this course is to learn the modern, provable security based approach to cryptography
- Introduce some of the core topics and some "hot areas"
- Learn the mathematical language used to express cryptographic concepts and **speak** this language

Course Objectives

- Cryptography is full of *seeming* paradoxes. To develop confidence in any designed scheme, it is important to *prove* its security. The primary goal of this course is to learn the modern, provable security based approach to cryptography
- Introduce some of the core topics and some "hot areas"
- Learn the mathematical language used to express cryptographic concepts and **speak** this language
- Think intuitively but write rigorous proofs

Course Objectives

- Cryptography is full of *seeming* paradoxes. To develop confidence in any designed scheme, it is important to *prove* its security. The primary goal of this course is to learn the modern, provable security based approach to cryptography
- Introduce some of the core topics and some "hot areas"
- Learn the mathematical language used to express cryptographic concepts and **speak** this language
- Think intuitively but write rigorous proofs
- Students encouraged to conjecture

Course Objectives

- Cryptography is full of *seeming* paradoxes. To develop confidence in any designed scheme, it is important to *prove* its security. The primary goal of this course is to learn the modern, provable security based approach to cryptography
- Introduce some of the core topics and some "hot areas"
- Learn the mathematical language used to express cryptographic concepts and **speak** this language
- Think intuitively but write rigorous proofs
- Students encouraged to conjecture

Grand aim: Initiate into state-of-the-art research in Cryptography

Pre-requisites

No background in Cryptography is necessary. However, the following are expected:

Pre-requisites

No background in Cryptography is necessary. However, the following are expected:

- Familiarity with modular arithmetic, basic discrete probability and simple combinatorics. In general, some mathematical maturity in definitions and basic proof techniques is expected.

Pre-requisites

No background in Cryptography is necessary. However, the following are expected:

- Familiarity with modular arithmetic, basic discrete probability and simple combinatorics. In general, some mathematical maturity in definitions and basic proof techniques is expected.
- Basic familiarity with asymptotic (Big-O) notation, **P** & **NP** complexity classes, Turing machines

Pre-requisites

No background in Cryptography is necessary. However, the following are expected:

- Familiarity with modular arithmetic, basic discrete probability and simple combinatorics. In general, some mathematical maturity in definitions and basic proof techniques is expected.
- Basic familiarity with asymptotic (Big-O) notation, **P** & **NP** complexity classes, Turing machines
- If you have taken basic undergraduate math courses (such as discrete mathematics) involving proofs and undergraduate theory of computation/algorithms, you should do fine.

Pre-requisites

No background in Cryptography is necessary. However, the following are expected:

- Familiarity with modular arithmetic, basic discrete probability and simple combinatorics. In general, some mathematical maturity in definitions and basic proof techniques is expected.
- Basic familiarity with asymptotic (Big-O) notation, **P** & **NP** complexity classes, Turing machines
- If you have taken basic undergraduate math courses (such as discrete mathematics) involving proofs and undergraduate theory of computation/algorithms, you should do fine.
- For a refresh: We will briefly review *some* of the above today. Chapter 0 from The Joy of Cryptography (see link on course website) is *required reading*.

General Information

- **Course website:** `https://github.com/aarushigoel/ModernCryptography-Fall2020/wiki`
- **Instructor:** Abhishek Jain (AJ), `abhishek@cs.jhu.edu`
- **Instructor Office Hours:** Tuesdays 11am-12pm via Zoom
- **Teaching Assistant:** Aarushi Goel, `agoel10@jhu.edu`
- **TA Office Hours:** Thursday, 4-5pm
- **Course Assistant:** Matthew Lombardo, `mlombar7@jhu.edu`

Grading

- **Homeworks:** 9 HW assignments, each counts 5%, total 45%. Submission via Gradescope (see course website).
- Late homework submission: HWs that are 0-24 hours late will lose **HALF** of their value. HWs submitted more than 24 hours late carry no value at all.
- **Mid-term:** 20% (Tentative Date: Oct 19; may be changed.)
- **Final:** 25%
- **Class participation:** 10%

Collaboration

- You can collaborate with other students on homework problems (but not midterm and final exam!)
- However: you must write the solutions in your own words. Copy-pasted solutions will forfeit all points.
- You must also list the names of students you collaborated with for each problem
- Do not collaborate with more than 2 students.

Plagiarism

Plagiarism will be dealt with strictly. You will be IMMEDIATELY reported.

If you have a problem, talk to me. Do NOT cheat!
(Copying solutions from online sources counts as cheating!)

How to use the course

- **Grades:** Do well in homeworks & exams
- **Research:**
  - Solve extra-credit questions
  - Read additional prescribed material
  - Discuss with me
  - Target: find a topic you are interested in

Textbook

- No required or prescribed textbook.
- Class lectures and notes will serve as main study material. Will be available on class website.
- Look for suggestions on class website for supplementary online reading material and books.

Syllabus

The main (basic & advanced) topics we will cover:

- One-Time Pads and provable-security approach
- Computational Intractability
- One way functions
- Pseudo-randomness
- Key Agreement
- Symmetric Encryption
- Public-Key Encryption
- Hash Functions & Digital Signatures
- Zero-Knowledge Proofs
- Secure Multiparty Computation

Syllabus continued . . .

Some emerging areas we will discuss (time permitting):

- Differential Privacy
- Secure Machine Learning
- Quantum Cryptography
- ...

Review

1. Probability
2. Complexity Theory
3. Quantification

Part I: Probability Theory

Probability

- **Sample Space:** Sample space of a probabilistic experiment is the set of all possible outcomes of the experiment.
  - We will only consider finite sample spaces
  - In most cases, the sample space will be the set $\{0,1\}^k$ of size 2^k
- **Event:** Event is a subset of the sample space.

Probability

- **Sample Space:** Sample space of a probabilistic experiment is the set of all possible outcomes of the experiment.
  - We will only consider finite sample spaces
  - In most cases, the sample space will be the set $\{0,1\}^k$ of size 2^k
- **Event:** Event is a subset of the sample space.
- **Union Bound:** If S is a sample space and $A, A' \subseteq S$, then the probability that either A or A' occurs is
  $\Pr[A \cup A'] \leqslant \Pr[A] + \Pr[A']$

Probability

- **Sample Space:** Sample space of a probabilistic experiment is the set of all possible outcomes of the experiment.
  - We will only consider finite sample spaces
  - In most cases, the sample space will be the set $\{0,1\}^k$ of size 2^k
- **Event:** Event is a subset of the sample space.
- **Union Bound:** If S is a sample space and $A, A' \subseteq S$, then the probability that either A or A' occurs is
  $\Pr[A \cup A'] \leqslant \Pr[A] + \Pr[A']$

Example: Rolling a Die

Probability of either rolling an odd number or a mulitple of 3.

Sample Space: $\{1,2,3,4,5,6\}$

Event A – rolling an odd number: $\{1,3,5\}$

Event A' – rolling a multiple of 3: $\{3,6\}$

$$\Pr[A \cup A'] \leqslant 3/6 + 2/6 = 5/6$$

Distributions

Distribution:

- D is a distribution over sample space $\mathcal{S}$ if for every $s \in \mathcal{S}$, it assigns probability p_s s.t. $\sum_s p_s = 1$.
- Drawing an element from the sample space $\mathcal{S}$ according to this probability distribution (i.e., when we draw element s with probability p_s) is called *random sampling from distribution* D.

Distributions

Distribution:

- D is a distribution over sample space $\mathcal{S}$ if for every $s \in \mathcal{S}$, it assigns probability p_s s.t. $\sum_s p_s = 1$.
- Drawing an element from the sample space $\mathcal{S}$ according to this probability distribution (i.e., when we draw element s with probability p_s) is called *random sampling from distribution* D.

Uniform Distribution:

- U is called the uniform distribution over sample space $\mathcal{S}$, if it assigns the *same* probability $p = 1/|\mathcal{S}|$ to every element $s \in \mathcal{S}$.
- The process of sampling an element (say s) from the uniform distribution is also called *sampling uniformly (and) at random* from the sample space $\mathcal{S}$ and is denoted as follows:

$$s \xleftarrow{\$} \mathcal{S}$$

Random Variables

Random Variables: A random variable is a function that maps elements of the sample space to another set. It assigns values to each of the experiment's outcomes.

Example

Suppose we sample uniformly at random from $\{0,1\}^3$. Let N be a random variable that denotes the number of $1s$ in the sampled string, i.e., for every $x \in \{0,1\}^3$, $N(x)$ is the number of $1s$ in x.

$$\Pr_{x \xleftarrow{\$} \{0,1\}^3}[N(x) = 2] = 3/8$$

Conditional Probability

Conditional Probability: The conditional probability of event B in relation to event A is the probability of event B occurring when we know that A has already occurred.

$$\Pr[B|A] = \Pr[A \cap B] / \Pr[A]$$

Example

Drawing Kings from a deck of cards.
Event A is drawing a King first, and Event B is drawing a King second.

$$\Pr[A] = \frac{4}{52}$$

$$\Pr[B|A] = \frac{3}{51}$$

Independent Events

We say that B is independent from A if $\Pr[B|A] = \Pr[B]$, i.e.,

$$\Pr[A \cap B] = \Pr[A].\Pr[B]$$

Example

Tossing a coin. The probability that heads shows up on two consecutive coin tosses,

$$\Pr[HH] = \Pr[H].\Pr[H] = \frac{1}{2}.\frac{1}{2} = 0.25$$

Each toss of a coin resulting in a heads is an Independent Event.

Pair-wise Independence

Pairwise Independent Random Variables: Let $X_1, X_2 \ldots\ldots X_n$ be random variables. We say that the X_i's are pairwise-independent if for every $i \neq j$ and all a and b, it holds that:

$$\Pr[X_i = a \text{ and} X_j = b] = \Pr[X_i = a].\Pr[X_j = b]$$

Example

We throw two dice. Let A be the event "the sum of the points is 7", B the event "die #1 came up 3" and C the event "die #2 came up 4".

$$\Pr[A] = \Pr[B] = \Pr[C] = \frac{1}{6}$$

$$\Pr[A \cap B] = \Pr[B \cap C] = \Pr[A \cap C] = \frac{1}{36}$$

$$\Pr[A \cap B \cap C] = \frac{1}{36} \neq \Pr[A].\Pr[B].\Pr[C]$$

A, B and C are pairwise independent but not independent as a triplet.

Part II: Complexity Theory

Algorithms and Turing Machines

- **Turing Machine:** A Turing machine is a hypothetical machine that can simulate any computer algorithm. It is equipped with a set of infinite tapes – a read-only *input* tape, read/write *work* tapes and an *output* tape – consisting of equal sized cells that contain symbols from some (prescribed) alphabet. All computations are performed by making changes to the contents on the work tapes, as described by a *transition function*. When the computation is completed, the machine writes the output on the output tape and *halts*.

Algorithms and Turing Machine (contd.)

- **Algorithm:** An algorithm is a Turing machine whose inputs and outputs are binary strings over the alphabet $\Sigma = \{0, 1\}$.
- **Randomized Algorithm:** A randomized algorithm, also called a *probabilistic Turing machine* is a Turing machine that is equipped with an extra randomness tape. Each bit of randomness tape is uniformly and independently chosen. The output of a randomized algorithm is a distribution.
- **Running Time:** The *running* time of an algorithm is measured by the number of steps taken by the Turing machine to complete the computation.

Running Time

An algorithm is said to run in time $T(n)$ if for all $x \in \{0, 1\}^n$ if the Turing machine halts within $T(|x|)$ steps.

Polynomial Time

Polynomial Time

The running time $T(n)$ of an algorithm is said to be polynomial, if there exists a constant c, such that $T(n) = n^c$.

- An algorithm is said to be *efficient* if it runs in polynomial time.
- A *probabilistic polynomial time* (PPT) algorithm is a randomized algorithm that runs in polynomial time.
- This notion captures what we can efficiently do *ourselves*.

Complexity Classes

- **Complexity Class P:** A language L is said to be in **P**, if there exists a polynomial-time algorithm which for any input x, can decide whether $x \in L$.
- **Complexity Class NP:** A language L is in **NP** if the following conditions hold:
  - There exists a Boolean relation $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$, such that $x \in L$ if and only if there exists a y such that $|y| \leqslant \mathsf{poly}(|x|)$ and $(x, y) \in R_L$.
  - There exists a polynomial-time algorithm which for any input (x, y) can decide whether $(x, y) \in R_L$.

Complexity Classes (contd.)

- **NP-Completeness:** A language is NP-complete if it is in NP and every language in NP is polynomially reducible to it.
  A language L is polynomially reducible to a language L' if there exists a polynomial-time-computable function f such that $x \in L$ if and only if $f(x) \in L'$.

Asymptotic Notations

- **Big-O:** The function $f(n)$ is $O(g(n))$ if $\exists$ constants c, n_0, such that $\forall n \geqslant n_0$

$$0 \leqslant f(n) \leqslant c.g(n)$$

- **Big-Omega:** The function $f(n)$ is $\Omega(g(n))$ if $\exists$ constants c, n_0, such that $\forall n \geqslant n_0$

$$0 \leqslant c.g(n) \leqslant f(n)$$

- **Theta:** The function $f(n)$ is $\Theta(g(n))$ if $\exists$ constants c_1, c_2, n_0, such that $\forall n \geqslant n_0$

$$0 \leqslant c_1.g(n) \leqslant f(n) \leqslant c_2.g(n)$$

Asymptotic Notations (contd.)

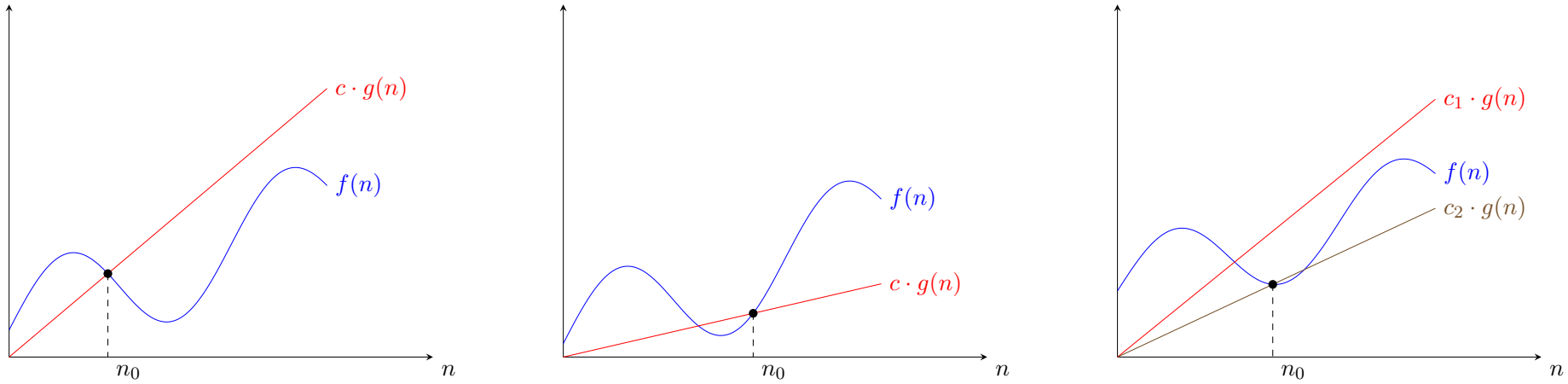

Figure: Examples of the O, Ω and Θ notations.

- **Small-o:** The function is $f(n)$ is $o(g(n))$ if $\forall$ constant c, $\exists$ constant n_0, such that $\forall n \geqslant n_0$

$$0 \leqslant f(n) < c.g(n)$$

- **Small-omega:** The function is $f(n)$ is $\omega(g(n))$ if $\forall$ constant c, $\exists$ constant n_0, such that $\forall n \geqslant n_0$

$$0 \leqslant c.g(n) < f(n)$$

Part III: Quantification

Quantification

- **Universal Quantification:** The following quantifier indicates that for all x in the set A, the statement $P(x)$ is true.

$$\forall x \in A,\ P(x)$$

- **Existential Quantification:** The following quantifier indicates that there is at least one x in A such that the statement $P(x)$ is true.

$$\exists x \in A,\ P(x)$$

Nesting Quantifiers. We can nest the above quantifiers in an arbitrary manner. For instance, the following is a valid nesting

$$\forall x_1 \in A_1, \forall x_2 \in A_2, \exists x_3 \in A_3, \forall x_4 \in A_4\ P(x_1, x_2, x_3, x_4).$$

Order of Quantifiers

When the nested quantifiers are different, the order of quantifiers matters.

Example

Consider the two following statements

1. $\forall x \in \mathbb{Z}, \exists y \in \mathbb{Z}$ s.t. $x + y = 4$
2. $\exists x \in \mathbb{Z},$ s.t. $\forall y \in \mathbb{Z}\ x + y = 4$

The first statement is true, since for every fixed x, we can set y to be $4 - x$. Thus existence of such a $y \in \mathbb{Z}$ is guaranteed.
The second statement says that there must be a single x such that for every value of $y \in \mathbb{Z}$, $x + y = 4$. This statement cannot be true.

Negation of Quantifiers

- $\neg\left(\forall x \in A,\ P(x)\right) \iff \exists x \in A,\ \neg P(x)$
- $\neg\left(\exists x \in A,\ P(x)\right) \iff \forall x \in A,\ \neg P(x)$
- $\neg\left(\exists x \in A, \forall y \in B,\ P(x,y)\right) \iff \forall x \in A, \exists y \in B,\ \neg P(x,y)$
- $\neg\left(\forall x \in A, \exists y \in B,\ P(x,y)\right) \iff \exists x \in A, \forall y \in B,\ \neg P(x,y)$

Here, $\iff$ is the notation for *if and only if* that connects two statements, where either both statements are true or both are false.